uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática

2019 - 2020

Trabajo Fin de Grado

"DISEÑO E IMPLEMENTACIÓN DE UN PROCESO DEVOPS CON CONTROL DE CALIDAD Y SEGURIDAD DEL CÓDIGO"

Esther Antón Durández

Tutor

José María Álvarez Rodríguez

Colmenarejo, 2020

RESUMEN

La presente memoria se trata de la documentación de lo que supone el diseño y la implementación de un sistema local de integración continua. Este es un concepto que durante los últimos años ha ido cogiendo fuerza en la industria del desarrollo de software. Se explica qué es DevOps, por qué toma fuerza con respecto a los antiguos modelos de deslocalizar los equipos de desarrollo de los equipos de operaciones, prácticamente unificándolos. En este documento es descrito el proceso de creación de un entorno de desarrollo e integración continua en local, y el diseño del proceso DevOps creado. Se describirán las ventajas de la metodología, así como los pasos que se han seguido para la configuración del sistema y su uso.

Palabras clave: Integración continua, DevOps, pipeline, jobs, despliegues, métricas, calidad, seguridad, Jenkins, SonarQube, análisis, repositorio remoto.

AGRADECIMIENTOS

A mi familia, porque no soy modélica pero el amor por la familia puede con todo. Gracias por no tirar la toalla conmigo, incluso cuando quería tirarla yo, porque mirad dónde estamos ahora. Gracias por enseñarme el valor de las cosas y del esfuerzo. Sólo espero que os sintáis orgullosos, como yo.

A mi tutor José María, porque di con él de forma fortuita y con su ayuda conseguí encontrar un tema que realmente me gustaba para poder poner el broche a esta carrera de fondo.

A los amigos de siempre, que durante este viaje han sabido tanto animarme como espabilarme cuando más lo necesitaba. A los dos que conocí el primer año de carrera, con los que he estado desde entonces y que ya son familia. Sin el aliento que nos hemos dado entre nosotros, llegar a este día habría sido imposible. Gracias.

A ti, Dani. Por serlo todo. Parte de mi carrera es tuya. Por todo lo que vendrá. Por todo lo que hemos conseguido, y por lo que trabajamos cada día para tenerlo. Te quiero. Gracias no es suficiente.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

INDICE DE TABLAS

1. INTRODUCCIÓN: MOTIVACIÓN Y OBJETIVOS

En este primer capítulo del documento que constituye el trabajo de fin de estudios, se hace una breve puesta en contexto tanto al proyecto como a la memoria.

En primer lugar, se describe la motivación que lleva a querer realizar el proyecto acerca de esta temática, los objetivos que se intentan lograr en él, y por último la estructura de la memoria, describiendo, sin mucho detalle, el contenido de cada uno de los apartados.

1.1. Motivación

Actualmente, y desde hace algo más de un año y medio, formo parte de un equipo de desarrollo de las herramientas corporativas de un gran cliente del mundo de las telecomunicaciones. Una de las características del proyecto y del equipo es que los integrantes no son únicamente desarrolladores o analistas, sino que también actúan como el equipo de operaciones.

En el último cuarto de 2019, se empieza a trabajar con el equipo de DevOps del cliente, por lo que se comienza a utilizar su metodología y herramientas para el control de versiones de los desarrollos, los despliegues, las integraciones de los desarrollos, y las métricas de calidad sobre el código fuente de las aplicaciones.

Esta situación lleva a querer indagar más acerca de DevOps, las diferentes herramientas, y los conceptos de *continuous integration* (integración continua) y *continuous delivery* (entrega continua):

- En primer lugar, conocer las herramientas que se empiezan a utilizar como parte del día a día del proyecto, a nivel profesional. Las posibilidades que estas ofrecen, así como encontrar los motivos que llevaban al equipo de DevOps a convencer a los desarrolladores de que adaptarse a esta metodología no es sólo beneficioso a nivel de negocio, sino que también resulta mejor a nivel de desarrollo y entrega.

- Además de esto, estudiar los beneficios que pueden traer el diseño e instalación de una réplica local de este sistema de integración continua, para que todos los miembros del equipo, encargados o no de los despliegues de los componentes, conozcan y usen diariamente la nueva filosofía con la que se empieza a trabajar.

Por lo que, juntando esta nueva situación y la curiosidad por el mundo DevOps, lleva a querer hacer un pequeño sistema local que pueda ayudar en las tareas de desarrollo de componentes del día a día del equipo, tanto de los miembros actuales como de los miembros futuros.

1.2. Objetivos

Vista la motivación que impulsa la realización de este proyecto, cabe fijarse unos objetivos que lleven a hacer un trabajo de calidad.

Los principales objetivos son los siguientes:

- **Diseñar e implementar en local un sistema DevOps.** En lo medida de lo posible, similar al sistema que se utiliza en el proyecto empresarial mencionado anteriormente.
- **Investigar acerca de DevOps**, y así poder adoptar esta metodología cuyo uso cada vez está más extendido en el mundo de la producción de software.

Otros objetivos, derivados de los dos principales son:

- Crear un sistema local que sea multiplataforma. Es decir, que pueda ser instalado en cualquier ordenador, independientemente del sistema operativo de éste.
- Estudiar y medir la reducción de tiempos desde que se desarrolla un componente hasta que puede ser desplegado en un servidor, sin importar que este sea local o remoto.
- Generar una metodología de trabajo lo más estandarizada posible, que pueda ser aplicada independientemente del proyecto en el que se esté trabajando.
- Llegar, si fuera posible, a crear una herramienta que pueda ser explotada por cualquier desarrollador, de cualquier nivel, o incluso contribuir a que en un futuro próximo los alumnos de la Universidad Carlos III de Madrid (UC3M) puedan montarse un sistema DevOps para el desarrollo de sus prácticas de las distintas asignaturas.

1.3. Estructura

Esta memoria consta de los siguientes apartados:

- **Capítulo 1: Introducción. Motivación y objetivos.** Como se puede leer anteriormente, en este capítulo se explica la motivación que lleva a la realización del proyecto, los objetivos de éste, y la estructura del documento de la memoria.
- **Capítulo 2: Estado del arte.** Lo referido a la explicación del concepto de DevOps, o a la integración continua. Las diferencias y aplicaciones que tiene con las metodologías ágiles. Se refieren las tecnologías que conforman el sistema, y se analizan con respecto a otras opciones que pueden encontrarse actualmente en el mercado.
- **Capítulo 3: Análisis del sistema.** Se estudia el sistema que se crea con este proyecto. Cuáles son las capacidades del sistema, los requisitos y se representan diferentes casos de uso.
- **Capítulo 4: Diseño del sistema.** Se explica cómo se alcanza la solución presentada. Se entra a detallar las tecnologías por las que se opta para establecer el sistema.
- **Capítulo 5: Implementación y pruebas.** A modo de guía, se detallan los pasos seguidos para desarrollar e inicializar el entorno local. Desde la configuración e instalación de los componentes, a los pasos a seguir para la utilización de éste. En el apartado de pruebas, se muestran y analizan las pruebas llevadas a cabo en el sistema para contrastar el correcto funcionamiento de éste.
- **Capítulo 6: Planificación y presupuesto.** Con la realización de un diagrama de Gantt se muestra la planificación que se sigue para la realización del proyecto. Se desglosa el presupuesto de los costes directos e indirectos derivados de la realización del trabajo.
- **Capítulo 7: Marco legal y entorno socio-económico.** Trata de la legislación aplicable al ámbito de DevOps, y a este sistema en concreto. Las implicaciones legales que puede traer, licencia de los datos analizados en el sistema, y licencias de las distintas herramientas que lo componen. En cuanto al marco socio-económico, se argumentan los beneficios que reporta la mejora de procesos para proveer mejores servicios o productos.
- **Capítulo 8: Conclusiones y trabajo futuro.** Es un último capítulo enfocado a las conclusiones a las que se llega tras crear el sistema, y las mejoras que pueden realizarse para desarrollar un sistema aun más completo.

- **Bibliografía.** Las referencias consultadas para diseñar el sistema y la generación de este documento.
- **Diccionario.** Guía de términos y acrónimos utilizados a lo largo del documento.

2. ESTADO DEL ARTE

En este apartado de la memoria se explican los conceptos asociados a este proyecto. Se listan tecnologías disponibles actualmente en el mercado, y se contextualiza este trabajo.

2.1. DevOps

DevOps es la combinación de diversas culturas, prácticas, filosofías y metodologías que potencian e incrementan la capacidad de los equipos de producir servicios y aplicaciones de mayor calidad, y en menor tiempo (Microsoft Azure, s.f.).

El término DevOps surge de dos términos en inglés que son desarrollo y operaciones (Development / Operations). Los equipos que trabajan bajo un modelo DevOps tienen la característica de que, al contrario que aquellos que generalmente trabajan de forma aislada como son los equipos de desarrollo, u operaciones de infraestructura, ahora trabajan de forma coordinada y colaborativa. De esta manera, se observa que los productos que se entregan son mejores, se realizan en menores tiempos, y son productos o servicios más confiables.

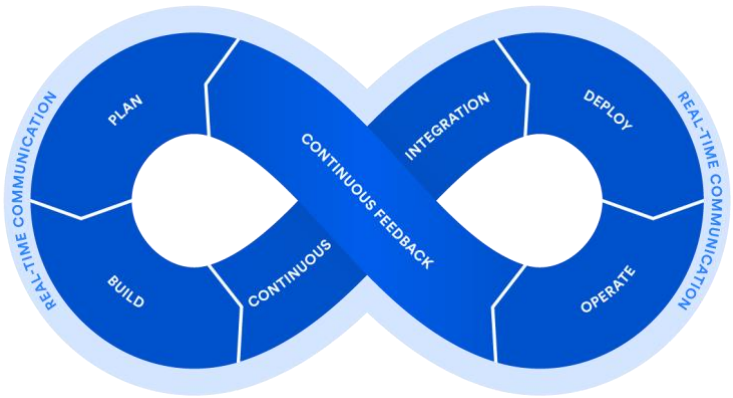


Ilustración 1 – Bucle ciclo de vida DevOps (Quijano, 2018)

La aplicación de esta filosofía, o método, proporciona una velocidad mayor a las organizaciones para ofrecer productos, aplicaciones, o servicios, haciéndolas más competitivas en el mercado.

Los equipos, anteriormente mencionados, junto con los equipos de seguridad y control de calidad, ahora colaboran de manera coordinada durante todo el ciclo de vida del producto. Con

DevOps, estos equipos automatizan procesos que antes se hacían de manera manual y más tediosa, como pueden ser, por ejemplo, los test unitarios o los despliegues de los componentes.

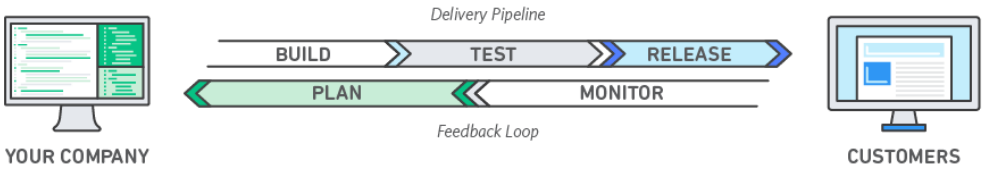


Ilustración 2 - Delivery Pipeline – Ciclo de Feedback (AWS, s.f.)

2.1.1. Fases del ciclo de vida de un proyecto con DevOps

Como se puede ver en la Ilustración 1, el ciclo de vida de un proyecto aplicando la filosofía DevOps contempla 6 fases diferenciadas que se repiten de manera iterativa durante la vida de dicho proyecto (Juan Quijano, 2018). Se explica, a continuación, en qué consiste cada uno de estos pasos:

Planning **(**Planificación**):** En este primer paso, se identifican y priorizan los requisitos que debe cumplir el servicio, o producto, que se desarrolla en el proyecto. Se identifican también cuales son las tareas que se deben realizar, y en qué orden, para luego poder llevarlas a cabo según sean más relevantes para el usuario final.

Fase de construcción (*Build*): En este paso, los requisitos ya están identificados, y se comienza el desarrollo del código del producto. Como bien indica el nombre de la fase, es el momento en el que empieza a construirse el software.

Fase de integración continua (Continuous Integration): Esta etapa viene de la mano de la fase de construcción. En esta fase, que es conveniente realizar de forma periódica e incluso diaria, se integra el código recientemente construido con el ya existente, así como el código nuevo con otro desarrollado por el resto de miembros del equipo. En esta fase, se compila todo el código, se unifica en un repositorio conjunto, y se realizan las pruebas de calidad y seguridad sobre los desarrollos.

Despliegue (*Deploy*): Una de las muchas ventajas de utilizar esta filosofía es que pueden automatizarse o programarse los despliegues de los nuevos componentes. No solo habla esta fase de los despliegues del software en un entorno de producción, sino que suelen existir dos entornos previos al entorno final. El primer entorno, comúnmente llamado entorno de integración, es un entorno de pruebas en el que suelen realizarse despliegues para hacer pruebas funcionales de los desarrollos una vez se han integrado con el código previo. Otro entorno es el entorno de

preproducción o pre productivo. Este entorno es más parecido a producción, donde pueden hacerse pruebas más completas, o realistas, de los desarrollos, antes de hacer un despliegue en producción. Esto ayuda a detectar y reducir los posibles errores que puedan darse en el entorno productivo una vez realizado el despliegue para los usuarios finales.

Monitorizar (*Operate*): La traducción literal de esta fase sería "operar" o "actuar". En esta fase se sigue el rendimiento de las nuevas funcionalidades. No sólo el rendimiento en cuanto a mediciones de memoria física, tiempos de carga, etc. Sino monitorización de las incidencias que puedan surgir una vez que se ha desplegado el software. Este tipo de incidencias son las relacionadas con posibles conflictos entre componentes, o dependencias sin resolver.

Feedback (realimentación/reacción) **continuo (*Continuous feedback*):** Por último, antes de comenzar de nuevo el ciclo, es importante tener presente que hay mantener abiertas vías de comunicación con todas las personas interesadas del proyecto para recibir sus opiniones acerca de las nuevas funcionalidades. De esta manera, abordar de nuevo el plan para futuras mejoras o modificaciones sobre el sistema.

2.1.2. Beneficios de aplicar DevOps

Como se menciona anteriormente, la aplicación de la filosofía DevOps trae consigo diversos beneficios. Se explican varios de estos, a continuación:

Confiablidad: La calidad y seguridad del código aumenta con este modelo de trabajo. Los evolutivos y las entregas se desarrollan de forma más rápida, consiguiendo así que la percepción del cliente y usuario final sea más positiva. Esto se hace mediante el uso de prácticas como pueden ser la integración continua (CI – Continuous Integration) o la entrega continua (CD – Continuos Delivery), que permiten comprobar que cada actualización realizada sobre el producto es funcional, válida y segura gracias, no solo al feedback de desplegar los productos sino, al feedback instantáneo de las herramientas de calidad. Otras prácticas como la monitorización y análisis, para el desarrollo del consiguiente plan de actualización, permiten también mantener la información de la acogida de la nueva funcionalidad a tiempo real.

Velocidad: Otro beneficio notable es la velocidad a la que los equipos de trabajo pueden realizar entregas de los productos. No solo es notable en las entregas, sino que trabajar con esta filosofía produce que, como proveedor de una aplicación o servicio, se pueda amoldar a las

tendencias del mercado y evolucionar los productos de forma más rápida, consiguiendo así una mayor satisfacción por parte del cliente y del usuario final. Aumenta la velocidad también en el ámbito de la reparación de posibles *bugs* (fallos, o incidencias) que puedan producirse.

Escalabilidad: De los dos anteriores beneficios deriva este. Permite dimensionar el servicio, o aplicación, sin riesgo de perder seguridad o calidad, sin ralentizar los desarrollos. Todo esto incrementando la complejidad del producto. No importa la dimensión del proyecto, la filosofía DevOps siempre es aplicable y agiliza el desarrollo desde el comienzo.

2.2. Integración continua

Se puede entender la integración continua como una práctica particular de desarrollo de productos de software recomendada en la filosofía DevOps *(SolutionDot, 2018)*. Antes de la aplicación de estas metodologías, lo más generalizado era que los desarrolladores del software trabajaran aislados, y una vez se terminara el desarrollo del producto, combinaran los distintos códigos en una única versión del producto. El intento de juntar todos los cambios de código en la fase previa a la puesta en producción se traduce en una tarea complicada y ardua, y un mayor número de errores en los productos.

Ilustración 3 - Integración continua - Entrega continua (AWS, s.f.)

> *"La integración continua se refiere a la fase de creación y pruebas de unidad del proceso de publicación de software. Cada revisión enviada activa automáticamente la creación y las pruebas." (AWS, s.f.)*

De los problemas anteriormente descritos surge la necesidad de subsanarlos. Con la integración continua, como puede empezar a intuirse por su nombre, los desarrollos del equipo se combinan cada poco tiempo. Estos cambios se *mergean* (fusionan), o integran, normalmente en un repositorio del código compartido con control de versiones. Cuando un desarrollador integra los cambios de su

código con el del resto del equipo en ese repositorio remoto, normalmente se disparan las pruebas automatizadas que detectan la correcta integración de esos cambios sobre el código actual *(AWS, s.f.)*.

2.2.1. Beneficios de la integración continua

Resolución de errores: Al contar con una serie de pruebas automáticas en la integración de los cambios, la detección y consiguiente resolución de errores es mucho más inmediata y eficiente.

Entrega rápida: Permite que los equipos de desarrollo puedan hacer despliegues de funcionalidades con mayor frecuencia.

Productividad mayor: Los equipos que trabajan con integración continua experimentan una mejoría de la productividad, al liberar a los desarrolladores de tareas, como las pruebas o despliegues, para poder enfocarse en una mayor eficiencia o una mayor calidad del código.

2.3. DevOps y las metodologías ágiles

Aunque se trata de dos cosas distintas, se tiende a confundirlas porque ambas filosofías tienen algunas similitudes.

Estos conceptos, junto con 'ITIL', o 'SCRUM' por ejemplo, suelen mezclarse debido a que ambos están movidos por el paradigma 'LEAN'.

El paradigma, o metodología, 'LEAN' es un modelo que se basa en la mejora continua, así como en la tendencia a lo excelso y aportar un mayor valor al cliente final. Estos son los ejes fundamentales de este modelo.

Para identificar y diferenciar DevOps de "*agile*" bajo este modelo, puede hacerse teniendo en cuenta que *agile* o *scrum* están enfocadas al desarrollo del software como tal, y que DevOps se aplica para la integración del ciclo de vida de ese software. Ambas parten del modelo *lean*, pero no aplican las mismas técnicas o términos.

> *"Mucha gente piensa que "metodología ágil" es sinónimo de scrum y que DevOps significa "entrega continua". Esta simplificación excesiva genera una tensión innecesaria entre la metodología ágil y DevOps, pero te sorprendería saber que se llevan estupendamente." (BUCHANAN, 2020)*

Aunque se han relacionado anteriormente *agile* con el desarrollo de software, no es una metodología que sea solo aplicable al mundo IT (Tecnología de la información, de sus siglas en inglés). Las metodologías ágiles son también frecuentemente aplicables al ámbito de negocio.

Como se explica anteriormente, son metodologías que se basan en la mejora continua y un mayor valor al cliente o usuario final. Esto, enfocado a procesos de negocio o en otros ámbitos de una empresa, es igualmente aplicable.

Quedan atrás los modelos monolíticos de gestión en las compañías. Esta visión, junto con un equipo de desarrollo que trabaje con la filosofía DevOps elabora productos mucho más completos y robustos. Si se topara con un producto que necesita aplicar un cambio sobre los requisitos, por las observaciones que se han llevado a cabo por el equipo agile de la respuesta de los usuarios hacia el uso de un producto, puede resultar mucho más sencillo aplicarlos para el equipo de desarrollo.

2.4. Kubernetes

Es un sistema open source para automatizar los despliegues y manejar y escalar aplicaciones contenerizadas, es decir, un orquestador para contenedores *(Cosio, 2019)*. Ya relacionada con la parte del ciclo de operación y producción, no tanto con la de desarrollo. Estos contenedores portátiles y autosuficientes suelen basarse en la tecnología **Docker**, que pueden ejecutarse tanto de manera local como en la nube.

Sus características más importantes son:

- **Escalabilidad y balanceo:** Se refiere a la capacidad del clúster de crecer dentro de unos estándares definidos. La definición de escalabilidad de Kubernetes se basa en dos conceptos: Indicadores de nivel de servicio (SLI), y objetivos de nivel de servicio (SLO). Mientras los SLI son genéricos (definen qué y cómo medimos), los SLO ofrecen garantías específicas y satisfacerlos puede depender del cumplimiento de algunos requisitos específicos. Ej. Específicos: Configuración, o carga, del clúster. Y, el balanceo, es el servicio que distribuye las cargas en los recursos del clúster de Kubernetes. *(CLOUD, G., s.f.)*
- **Automatización de la administración de los contenedores:** Kubernetes es un orquestador de contenedores, y para administrarlos, aplica la automatización de procesos. Esto hace que resulte más fácil para los desarrolladores gestionar sus aplicaciones. Kubernetes maneja los contenedores, pero no los crea.

- **Facilidad de despliegue:** El uso de contenedores facilita el despliegue de aplicaciones al generar imágenes de un contenedor en lugar de una máquina virtual. Permite que la imagen de un contenedor se construya y despliegue de forma periódica y fiable, con un gran soporte en caso de necesitar una marcha atrás, ya que la imagen no cambia.

Cabe destacar que, dentro de la red local del equipo (localhost), se le otorga una IP al clúster de Kubernetes. Y a su vez, ese clúster tiene una red interna asociada. Esto proporciona versatilidad a la hora de configurarlo, y facilita la comunicación entre los diferentes servicios desplegados en él.

Contenedores

Un contenedor es una abstracción de un sistema operativo que permite la ejecución de aplicaciones en un entorno ligero y portable. La idea de un contenedor es muy similar a una máquina virtual, con la excepción de que no es necesario que arranque un sistema operativo al ejecutarse.

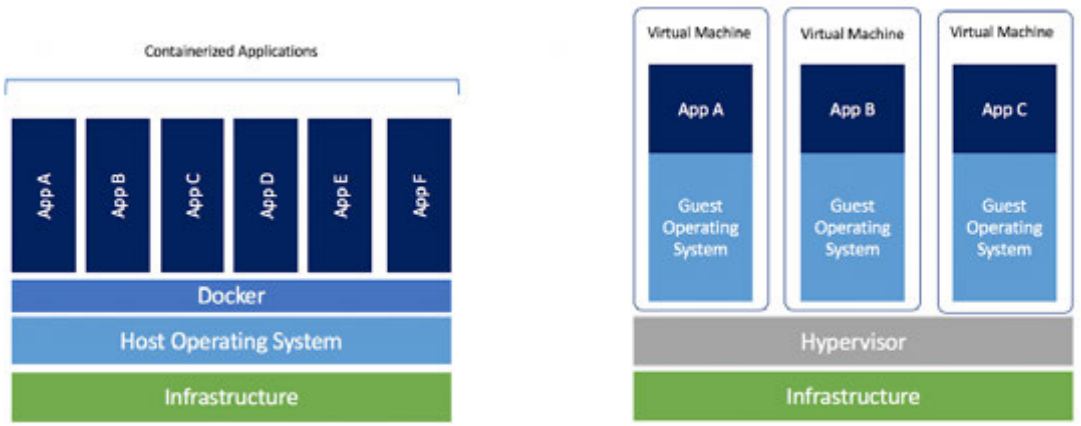


Ilustración 4 - Comparación de contenedores Docker con máquinas virtuales (Fong, 2018)

Esto posibilita el empaquetado de una aplicación en un contenedor junto con sus dependencias y, de esta manera, se podrá ejecutar en cualquier entorno.

Componentes

Los componentes básicos de un clúster de Kubernetes son:

- Pod: agrupación lógica de contenedores que comparten almacenamiento en disco, red y un comportamiento de ejecución. Cada Pod tiene asignado una dirección IP única dentro del clúster. Es la unidad mínima que administra Kubernetes, ya que no interacciona con contenedores directamente, solo con Pods.

- Servicio (*Service*): es una definición de un grupo de Pods y es utilizado para exponer dicho número de Pods a través de la red. Los principales tipos de servicios son:
    - **ClusterIP**: expone el servicio para su comunicación con otros componentes del clúster. Asigna una IP interna del clúster, por lo que no es accesible desde el exterior.
    - **NodePort**: expone el servicio en la dirección IP del clúster y en un puerto específico para acceder desde el exterior. Este tipo de servicio es el usado en este trabajo para acceder a las distintas aplicaciones desplegadas.
    - **Loadbalancer**: expone el servicio en una IP externa y distribuye el tráfico entre los Pods que componen el servicio. Para crear este servicio es necesario alojar el clúster en un proveedor de la nube (AWS, Google Cloud Platform, Azure).
- Volumen: un volumen es el espacio en disco en el que se alojan los archivos de los contenedores que forman los Pods.
- Nombre de espacio (*namespace*): agrupaciones lógicas de los distintos recursos que forman el clúster. Son usados principalmente para separar recursos en función de los usuarios o departamentos que utilizan el clúster.

2.5. Control de versiones

El control de versiones es cualquier tipo de práctica que monitoriza y proporciona control sobre los cambios en el código fuente. Los desarrolladores de software a veces utilizan software de control de revisiones para mantener la documentación y los archivos de configuración, además del código fuente.

Un software de control de versiones (CV) es un sistema que permite guardar un registro histórico de las modificaciones que se hacen sobre los archivos de modo que es posible recuperar versiones específicas en un futuro. *(Cañas, S., s.f.)*. Es una práctica muy extendida en el desarrollo de software, pero es aplicable para cualquier tarea que necesite un control minucioso de las modificaciones que sufre.

En un entorno colaborativo de desarrollo, es una herramienta esencial. Con varios miembros de un equipo modificando archivos del código fuente de un proyecto, se necesita hacer uso de un sistema de control de versiones.

Sus características principales son:

- **Manejo de conflictos:** Cuando son varios los desarrolladores que modifican un mismo archivo de código, pueden producirse conflictos al unificar los cambios. El control de versiones facilita la visión de los cambios que se han realizado, y la resolución del conflicto que pueda provocar.
- **Histórico del código fuente:** Tener un control de las versiones de los archivos, permite poder acceder a versiones previas estables del código de forma rápida.
- **Copias de seguridad del código:** Tener alojado el repositorio de código fuente de forma remota permite tener diversas copias de seguridad, quitando riesgo de pérdida de información.

Terminología común del control de versiones, que se utilizan a lo largo de este documento:

- **Rama**: Un conjunto de archivos bajo CV puede *ramificarse*, o *bifurcarse*, en un momento determinado para que, a partir de entonces, dos copias de esos archivos puedan desarrollarse a diferentes velocidades o diferentes formas, de forma independiente entre ellas.
- **Checkout**: Es la acción crear una copia de trabajo local desde el repositorio remoto. Cuando se hace una copia local desde el repositorio, el resto de desarrolladores no tiene acceso a las modificaciones que se realizan, hasta que sean publicadas.
- **Clone**: Clonar significa crear un repositorio que contenga las revisiones de otro repositorio. Puede entenderse como que dos repositorios son clones si están sincronizados y contienen lo mismo.
- **Repositorio**: Es el espacio donde se almacenan los archivos actualizados, y su histórico de cambios. Como norma general, está alojado en un servidor remoto.
- **Módulo:** Es un conjunto de archivos o directorios dentro de un repositorio, que conforma una funcionalidad concreta en el proyecto.
- **Commit:** Es la confirmación de un cambio. Es fusionar los cambios hechos en la copia de trabajo sobre el repositorio. Al confirmar un cambio, se genera una nueva versión del código sobre el repositorio.
- **Push:** Copiar la nueva versión del código del repositorio local, o copia de trabajo, sobre el repositorio remoto.
- **Pull:** Descargar, y fusionar, en la copia de trabajo local los cambios publicados en el repositorio remoto.

- **Merge:** Unificar, fusionar, o integrar. Se produce cuando se combinan dos o más cambios sobre un mismo archivo, o conjunto de archivos. Puede producirse tras un pull, normalmente de forma automática. Puede bifurcarse un mismo archivo, para que distintos desarrolladores lo modifiquen de forma paralela, y luego integrar los cambios para que en un solo archivo se unifiquen.

2.6. Métricas de calidad

La calidad del código puede medirse siguiendo distintos criterios, todos de forma subjetiva. Esto es así pues la calidad depende de la tecnología y del tipo de software que se desarrolle, ya que, por poner un ejemplo, la calidad del frontal de una tienda online no tiene nada que ver con la calidad de un producto de análisis de datos.

Dicho esto, la calidad del código se puede dividir en cinco bloques, o aspectos, si bien el peso de cada uno y la forma de calcularlo es relativo al producto que se esté desarrollando.

Mantenibilidad

La facilidad con la que un sistema o componente de software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno. (Sicilia, 2012).

En la mantenibilidad del software influyen factores como:

- Tamaño: si el código tiene un tamaño muy grande puede resultar difícil de manejar y modificar.
- Legibilidad: un código que se puede entender y se pueden identificar sus funcionalidades facilita su mejora.

Reusabilidad

Facilidad del código para volver a ser usado en otra implementación. Esto no implica necesariamente que el mismo código vaya a ser reutilizado para el mismo objetivo, sino que una porción de código puede ser adaptado para otra situación similar.

El uso de método o funciones (dependiendo del tipo de tecnología) beneficia a la reusabilidad del código, de igual manera que la herencia de clases y, en última instancia, la atomicidad del código.

Seguridad

La seguridad del código estático persigue minimizar el número de riesgos o amenazas que puede sufrir un producto de software.

Uno de los ejemplos más conocidos de riesgos de seguridad es el SQL Injection, que consiste en el uso de código SQL dónde no debería estar permitido realizar consultas. Otro riesgo es no utilizar contraseñas en el código sin cifrarlas, lo que puede incurrir en el robo de las credenciales. Estos son fallos de seguridad que pueden ser evitados realizando los controles pertinentes a la hora de escribir el código de las aplicaciones.

Fiabilidad

Según la norma ISO 25010, se define la fiabilidad de un sistema como la capacidad para desempeñar su función, cuando se usa bajo unas condiciones y periodo de tiempo determinados. A su vez, se divide en 4 características:

- Madurez: Satisface la necesidad para la que se ha programado en condiciones normales.
- Disponibilidad: La capacidad de estar operativo y accesible.
- Tolerancia a fallos: Su capacidad de estar disponible en presencia de otros fallos ajenos al componente.
- Capacidad de recuperación: Capacidad de recuperar datos y reestablecer su servicio en caso de fallo.

Portabilidad

Capacidad del código para ser utilizado en varios entornos o plataformas. Si bien esta característica puede ser muy dependiente del tipo de tecnología utilizada, la estructuración del código puede ayudar a que el software funcione con distintos componentes.

La atomicidad de las funcionalidades es un factor clave en la portabilidad ya que permite identificar la independencia de los distintos elementos que conforman un sistema.

2.7. Herramientas DevOps

Dentro de las distintas fases que conforman el ciclo de vida de desarrollo de un proyecto aplicando filosofía DevOps, pueden reconocerse diversas herramientas que se utilizan en cada una de ellas. Es posible encontrar que una misma herramienta es utilizada en distintas fases de este ciclo

de vida, dado que pueden tener funcionalidades que así lo permitan. A continuación, se explican algunas de estas herramientas y en qué fase del ciclo se aplican, siendo algunas de ellas las que han sido elegidas para la creación de nuestro sistema, y otras las más relevantes en el mercado actualmente (Moy, 2019).

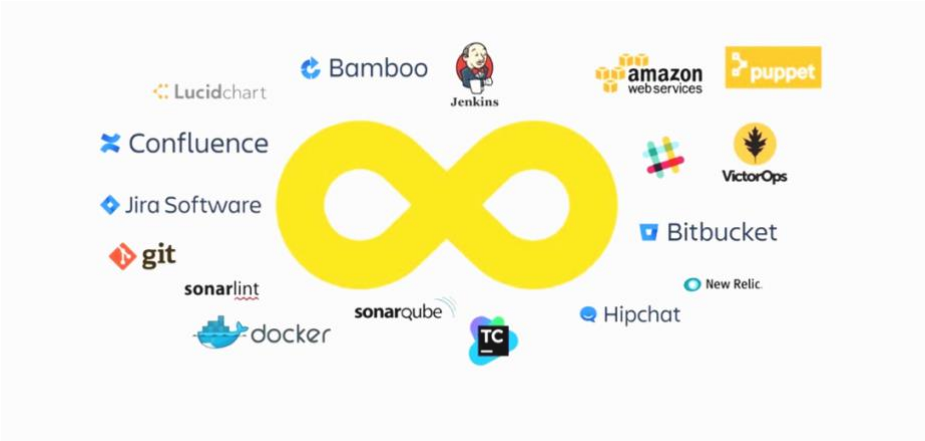


Ilustración 5 – Herramientas DevOps (Ochoa, 2018)

2.7.1. Jira (Gestión y planificación)

Para la primera fase del ciclo de planificación, Jira es una de las herramientas más populares y usadas del mercado. Se trata de una herramienta de seguimiento de tareas e incidencias de proyectos en línea. Es una herramienta muy visual que ayuda con la gestión del estado del desarrollo de los proyectos, así como la gestión de errores. Incorpora flujos de trabajo, y puedes integrarlo con Confluence, una herramienta del mismo desarrollador, que ayuda con la gestión de los requisitos del sistema durante el desarrollo. También puede integrarse con un repositorio de código para hacer trazabilidad entre tarea de un cambio y el propio cambio en el código, para un mejor seguimiento de la planificación.

2.7.2. Confluence (Documentación)

Referenciado en el apartado anterior, Confluence es un software que ayuda al trabajo en equipo, para que todos los integrantes estén alineados con los requisitos del sistema que se está desarrollando. Sirve tanto para definir requisitos en forma de historias de usuario, como para realizar documentación técnica de los desarrollos. De esta manera, se facilita la comunicación tanto con los usuarios como los miembros del equipo que desarrolla del proyecto.

2.7.3. Git (Control de versiones)

Para las fases que implican código, Git es un popular software de control de versiones. Imprescindible en las fases de integración continua, sobre todo en aquellos proyectos en los que existen numerosos archivos con código fuente y varios desarrolladores trabajando sobre un mismo componente. Junto con otros otras herramientas, por ejemplo, para la realización de test unitarios, contribuye a una mayor eficiencia y fiabilidad en los desarrollos evolutivos de los productos.

2.7.4. Jenkins (Orquestador/Automatización)

Jenkins es un 'orquestador'. Es un software *open-source* (código abierto) que permite la automatización de las distintas etapas de vida del software a partir de su desarrollo, como la compilación y las pruebas. También automatiza el despliegue de componentes y múltiples otras acciones.

2.7.5. SonarQube (Control de calidad y seguridad)

Es una plataforma de código abierto que permite monitorizar la seguridad y calidad del código que se está integrando. Muy flexible con las reglas de calidad que puedes aplicar, permite indicar umbrales según para qué métrica de calidad y así poder evitar despliegues de componentes no suficientemente seguros, o permitirlos si no son desarrollos preparados para el entorno de producción.

2.7.6. ELK Stack

Para las fases de monitorización, está ganando en los últimos tiempos en popularidad ELK. Son las siglas que combinan tres herramientas open source para una monitorización más eficaz de los productos. **Elasticsearch**, **Logstash** y **Kibana**. Con Elasticsearch como motor de búsqueda y indexador de *logs* (registros), Logstash como pipeline de ingesta de los registros, y Kibana como herramienta para la visualización de toda esa información de forma gráfica.

2.8. Tabla comparativa de herramientas para la solución adoptada

A continuación, se muestra una tabla comparativa donde se indican varias herramientas para cada etapa del ciclo de vida DevOps, y los tipos de licencias que tiene cada una de ellas, que resulta un factor determinante, así como poder ejecutarlo de forma local, en la elección de herramientas

para la realización del proyecto. Se crea la tabla tomando como referencia una tabla periódica que encontrada online muy ilustrativa con las actuales herramientas que más se usan (DevOps Practitioners, s.f.).

Elementos que componen la tabla:

X – Aplica en esta etapa.

Pago – Licencia de pago para el uso de la herramienta.

Freemium – Que la herramienta ofrece la versión básica con funcionalidades de manera gratuita, y funcionalidades adicionales en una versión de pago.

Open Source – El software open source, o de código abierto o licencia libre, implica que no solo es gratis, sino que su código fuente es accesible al público.

	Planificación	*Build*	*IC*	*Deploy*	*Monitorización*	*Licencia*	*Host-based*
Jira	X					Pago	
Trello	X					Freemium	
Gitea			X			Open source	X
GitLab			X			Open source	X
Jenkins		X		X		Open source	
Bamboo		X		X		Pago	X
Kubernetes				X	X	Open source	X
SonarQube					X	Open source	X
ELK						Open source	

Tabla 1 - Tabla comparativa de herramientas

3. ANÁLISIS DEL SISTEMA

Para la elaboración un sistema DevOps local funcional y de calidad, es preciso llevar a cabo un análisis de las necesidades de dicho sistema y una completa documentación del proceso. Para la elaboración de esta documentación, en este trabajo de fin de estudios se sigue el formato de Especificación de Requisitos Software (ERS) según la última versión del IEEE-830, ya que es un formato conocido, y recientemente utilizado en otros proyectos realizados.

3.1. Descripción general

3.1.1. Perspectiva del producto

El producto proporciona al usuario un sistema DevOps totalmente funcional en local, para poder familiarizarse con la metodología y trabajar con un modelo del ciclo de vida del desarrollo de software más eficiente.

3.1.2. Alcance del software

En principio el sistema no recibe un nombre. Si finalmente se generara un producto derivado de este proyecto, queda pendiente de ser nombrado.

Como se ha comentado en el apartado anterior, el fin de la generación de este entorno DevOps local es la familiarización con esta metodología, además de realizar un análisis de calidad y seguridad de las distintas versiones del código fuente que se esté desarrollando.

El sistema permite, mediante el uso de un software de control de versiones, generar las ramas que sean necesarias para el usuario, y analizarlas todas por separado si se requiere.

El sistema no automatiza el despliegue de componentes en un servidor, ni local ni remoto.

Las ventajas derivadas del desarrollo de este proyecto son que los desarrolladores que todavía no han trabajado en un proyecto en el que se aplican técnicas de integración continua, o DevOps, puedan utilizarlo como forma de trabajo en el día a día. De esta manera, cuando el desarrollador tenga que trabajar en un proyecto de esas características, ya cuenta con experiencia y soltura en este ámbito. Son metodologías cada vez más populares, por lo que puede resultar un gran acelerador de capacidades para los usuarios que lo utilicen.

3.1.3. Funciones del producto

Como capacidades generales del sistema, destacan los siguientes componentes:

- Un clúster de Kubernetes: Kubernetes es un orquestador de contenedores sobre los que corren el resto de componentes del sistema local de integración *(RedHat, s.f.)*.
- Un software de control de versiones: Con él se pueden tener diferentes ramas con diferente código en cada una de ellas, sobre las que permite hacer diferentes análisis sobre las versiones de código registradas en ellas.
- Un orquestador, o servidor de automatización, que admite automatizar algunas partes del proceso de desarrollo de software, como puede ser el lanzamiento del análisis de calidad o seguridad del código cuando se modifica y sube nuevo código al repositorio de código.
- Una plataforma de análisis de código, donde poder evaluar y visualizar el impacto que tienen las modificaciones sobre el código en cuanto a calidad o seguridad, ya que cuenta con un panel de control amigable donde poder detectar rápidamente dicho impacto.

El sistema, pues, es un compuesto de estos elementos que permiten analizar el código de forma automática a la par que se está desarrollando dicho código.

3.1.4. Características de los Usuarios

El sistema está enfocado a un solo perfil de usuario, que es el desarrollador de cualquier tipo de software. Se enfoca de un usuario con perfil técnico, capaz de instalar el sistema, así como de configurarlo para sus necesidades con ayuda de este documento a modo de guía, y con capacidad de programar, independientemente del lenguaje de programación.

Rol	Capacidades y competencias
Desarrollador	- Instalación del entorno / sistema local de integración - Configuración del entorno para adaptarlo a sus necesidades de programación - Conocimientos en programación

Tabla 2 - Usuarios - Roles y competencias

3.1.5. Restricciones

El sistema puede ser instalado con independencia del sistema operativo que tenga el ordenador, ya que son componentes que soportan tanto Linux, como Windows o MacOS; los sistemas operativos más comunes actualmente.

El análisis de código es posible para los lenguajes de programación que están cubiertos actualmente por la plataforma elegida con este fin. Para algunos lenguajes, la cobertura solo está disponible en la versión de pago de la plataforma.

La dimensión y potencia del sistema en local está limitada por la capacidad de procesamiento que tenga el equipo donde se quiera instalar. Si se intenta instalar en un ordenador con poca potencia, deben adaptarse las capacidades del sistema de integración de forma que pueda ser ejecutado y sea funcional.

Del ciclo de vida de desarrollo de software con DevOps, este producto no abarca la parte de despliegue ni de feedback continuo, ni las herramientas de planificación, dado que no se ha configurado ningún servidor sobre el que desplegar las compilaciones, pero es posible hacerlo. Con respecto al *feedback* (realimentación) continuo; no se cuenta con ningún usuario final o cliente que sugiera evolutivos sobre el producto construido, pero sí se obtiene feedback de los análisis de calidad y seguridad, que proporciona la herramienta de análisis, para hacer mejoras sobre el código. Con respecto a la falta de la fase de planificación, se debe a que es un sistema enfocado a la interacción con el código, y no a un proyecto de software completo.

3.2. Requisitos del proyecto

En este apartado se especifican los requisitos surgidos del análisis de las capacidades del sistema, el proceso DevOps diseñado, y las necesidades identificadas tras la familiarización con sistemas de integración continua en otros proyectos en los que se ha participado.

3.2.1. Plantilla de requisitos

La plantilla elegida para la especificación de estos requisitos es la siguiente:

Identificador - Nombre	
Necesidad	
Prioridad	
Origen	
Descripción	

Tabla 3- Plantilla de especificación de requisitos

Cuyos campos y posibles valores se explican a continuación:

- **Identificador**: Es la referencia única, e inconfundible, de cada uno de los requisitos para que cada requisito sea inequívocamente identificado. Existen dos posibles formatos:
    - RDX-Y: Siendo estos los requisitos del proceso DevOps, la X haciendo referencia a la inicial de la fase del ciclo de DevOps con la que está relacionado, y la Y un número del 01 en adelante, aumentando en una unidad.
    - RSX-Y: Siendo estos los requisitos del sistema, la X haciendo referencia al tipo de requisito del sistema, y la numeración incrementará del mismo modo que los requisitos del proceso DevOps. Siempre desde el 01, aumentando en una unidad por cada nuevo requisito.
- **Nombre:** Título breve del requisito.
- **Necesidad:** También identificada como prioridad, indica la necesidad de dicho requisito para la funcionalidad de nuestro sistema. Pueden indicarse los siguientes valores:
    - **Esencial:** indica que es un requisito de desarrollo obligatorio para la creación del sistema.
    - **Deseable:** Indica que el desarrollo de dicho requisito se trata de una característica que completa la funcionalidad del sistema, pero es posible negociar su implementación.
    - **Opcional:** Es un requisito que puede plantearse, pero que el sistema sigue siendo igual de válido y funcional si no se desarrolla.

- **Prioridad:** Este valor determina el orden en el que debe desarrollarse el requisito respecto a los demás identificados.

  - **Alta:** Un requisito con prioridad alta debe ser implementado antes que los requisitos clasificados con prioridad media o baja.
  - **Media:** Se debe implementar antes que los clasificados con prioridad baja, pero después de aquellos con prioridad alta.
  - **Baja:** En caso de ser implementados, lo son tras aquellos con prioridad alta y media.
- **Origen:** Indica a partir de quién surge este requisito. En este caso, siempre es el equipo de análisis y desarrollo, dado que actúa tanto de cliente como de desarrollador.
- **Descripción:** Una explicación más extensa que el título, pero que corresponde con el contenido de este.

3.2.2. Tipos de requisitos

Dentro de los requisitos, hemos creado dos grupos principales de ellos; requisitos del proceso DevOps, y requisitos del sistema. Dentro de los requisitos del proceso Devops, se dividen en las fases del proceso DevOps con los que se relacionan. No se incluye la fase de monitorización porque, aunque cuando se desarrolla se detecta si un software no funciona correctamente, y por lo tanto se detecta una incidencia, no se especifican requisitos para esta fase del ciclo.

Se pueden visualizar estas tipologías en el siguiente gráfico:

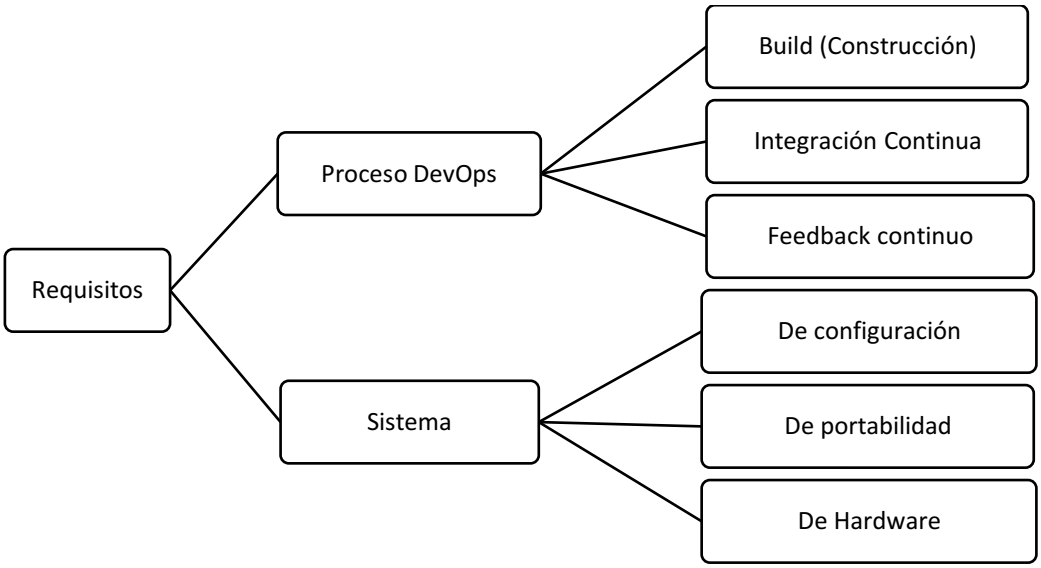


Ilustración 6 - Esquema de tipos de requisitos

3.2.3. Requisitos del proceso DevOps

Fase de Build (Construcción)

RDB-01 - Gestión de versiones de código	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	El desarrollador incluirá el código fuente de su aplicación en un repositorio de código accesible mediante GIT.

Tabla 4 - RDB-01 - Gestión de versiones de código

RDB -02 - Control de versiones multi rama	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	El proceso de DevOps analizará el código fuente de cualquier rama que se indique en el repositorio de código.

Tabla 5 - RDB -02 - Control de versiones multi rama

Fase de Integración Continua

RDIC-01 - Automatización del análisis del código	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	El proceso de DevOps lanzará automáticamente el análisis de código fuente cuando se realice un nuevo push.

Tabla 6 - RDIC-01 - Automatización del análisis del código

RDIC -02 - Automatización de tareas según la rama actual	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	El proceso de DevOps lanzará automáticamente diferentes tareas, si así se configura, cuando se realice un push en diferentes ramas.

Tabla 7 - RDIC -02 - Automatización de tareas según la rama actual

RDIC -03 - Compilación condicionada del código	
Necesidad	Media
Prioridad	Media
Origen	Analista
Descripción	El proceso DevOps podrá evitar automáticamente la compilación si las métricas de calidad sobre el código están por debajo de los umbrales fijados.

Tabla 8 - RDIC -03 - Tabla 8 - Compilación condicionada del código

Fase de Feedback Continuo

RDFC-01 - Análisis del código fuente	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	El proceso de DevOps analizará el código fuente alojado en un repositorio de código, generando métricas de calidad sobre dicho código fuente.

Tabla 9 - RDFC-01 - Análisis del código fuente

RDFC -02 – Umbrales de calidad	
Necesidad	Media
Prioridad	Media
Origen	Analista
Descripción	El proceso DevOps contará con una herramienta de análisis del código configurable para poder establecer el flujo del proceso en función del resultado del análisis del código.

Tabla 10 - RDFC -02 – Umbrales de calidad

RDFC -03 - Visualización de resultados de análisis	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	El proceso reflejará el resultado del análisis de código en el panel de control de la herramienta de análisis para poder consultarlo.

Tabla 11 - RDFC -03 - Visualización de resultados de análisis

RDFC -04 - Lenguaje de código analizado	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	El proceso podrá realizar análisis sobre el código cuando sea un lenguaje soportado por la herramienta de análisis.

Tabla 12 - RDFC -04 - Lenguaje de código analizado

RDFC -05 – Sugerencias de solución a incidencias	
Necesidad	Media
Prioridad	Baja
Origen	Analista
Descripción	El proceso mostrará sugerencias para resolver cuestiones que se encuentren tras el análisis del código del repositorio.

Tabla 13 - RDFC -05 – Sugerencias de solución a incidencias

3.2.1. Requisitos del sistema

Requisitos de configuración

RSC-01 - Iniciar el sistema	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	El usuario podrá iniciar el sistema mediante un único comando desde la consola del equipo en el que está instalado.

Tabla 14 - RSC-01 - Iniciar el sistema

RSC-02 - Configuración del sistema	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	El sistema permitirá ser configurado de forma amigable, para poder adaptarlo a las distintas necesidades de los desarrollos del usuario que lo utilice.

Tabla 15 - RSC-02 - Configuración del sistema

RSC-03 - Apagado del sistema	
Necesidad	Alta
Prioridad	Baja
Origen	Analista
Descripción	El usuario podrá parar el sistema mediante un único comando desde la consola del equipo en el que está instalado.

Tabla 16 - RSC-03 - Apagado del sistema

RSC-04 - Usuario del sistema	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	Sólo existirá un usuario y no requiere autenticación en el sistema

Tabla 17 - RSC-04 - Usuario del sistema

Requisitos de portabilidad

RSP-01 - El sistema correrá en una máquina virtual	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	El sistema se levantará sobre Minikube, una máquina virtual que se ejecuta sobre un hipervisor desplegando en ella un clúster de Kubernetes.

Tabla 18 - RSP-01 - El sistema correrá en una máquina virtual

RSP-02 - El sistema podrá instalarse en cualquier sistema operativo	
Necesidad	Alta
Prioridad	Alta
Origen	Analista
Descripción	Minikube permite desplegar un clúster de Kubernetes independientemente del sistema operativo que tenga el equipo.

Tabla 19 - RSP-02 - El sistema podrá instalarse en cualquier sistema operativo

Requisitos de hardware

RSH-01 - Memoria RAM del ordenador	
Necesidad	Media
Prioridad	Media
Origen	Analista
Descripción	El equipo en el que se instala el sistema debe disponer de un mínimo de 8GB de memoria RAM para un funcionamiento óptimo de las capacidades del sistema.

Tabla 20 - RSH-01 - Memoria RAM del ordenador

RSH-02 - Procesador del ordenador	
Necesidad	Media
Prioridad	Media
Origen	Analista
Descripción	El equipo en el que se instala el sistema debe disponer de un procesador Intel i5, o de mayor potencia para un funcionamiento óptimo de las capacidades del sistema.

Tabla 21 - RSH-02 - Procesador del ordenador

3.3. Presentación de los casos de uso

En este epígrafe se muestran tanto un diagrama de casos de uso identificados, como una explicación en formato expandido de cada uno de ellos.

3.3.1. Diagrama de casos de uso

En el siguiente diagrama pueden identificarse de forma gráfica cuáles son los casos de uso que encontrados tras el diseño del sistema:

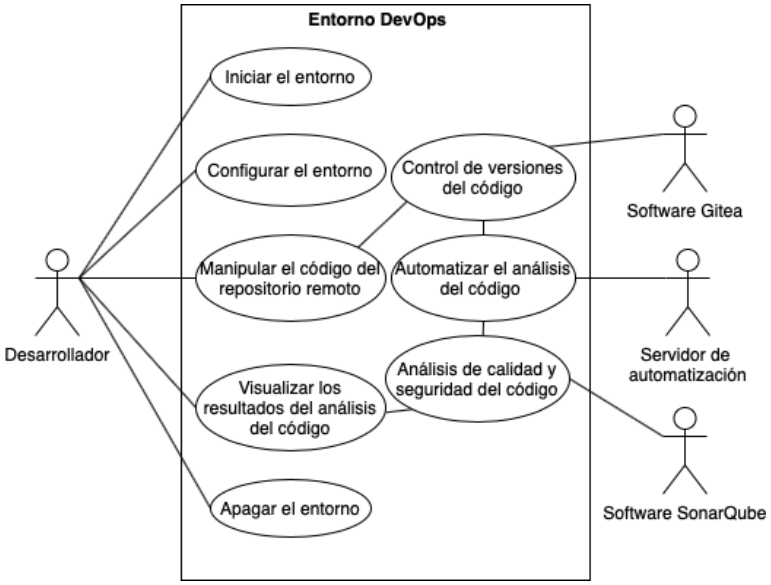


Ilustración 7 - Diagrama de casos de uso

3.3.2. Casos de uso en formato expandido

Para explicar los casos de uso en formato expandido, se hará mediante el uso de la siguiente tabla, ya que se ha utilizado con anterioridad en otros proyectos.

Caso de uso	***CU-XX***	
Actores		
Propósito		
Visión general		
Tipo		
Precondición		
Pos condición		
Desarrollo normal	**ACTOR**	**SISTEMA**
Desarrollo alternativo	**ACTOR**	**SISTEMA**

Tabla 22 - Plantilla de casos de uso en formato expandido

Siendo cada uno de los campos lo siguiente:

- **Caso de Uso**: Es la denominación inequívoca de cada caso de uso. CU seguido de una cifra, empezando a numerarlos desde el 01. Cuenta con una breve descripción, correspondiente a lo que aparece en el diagrama mostrado anteriormente.
- **Actores**: Personajes o sistemas que intervienen sobre el caso de uso concreto.
- **Propósito**: El fin que cumple el caso de uso.
- **Visión general**: La descripción en detalle del caso de uso.
- **Tipo**: Describe la necesidad de la realización del caso de uso. Pueden ser:
    - Primario: Representan procesos comunes más importantes del sistema.
    - Secundario: Representan procesos más raros, menos importantes.
    - Terciario: También dichos "opcionales". Procesos que pueden no abordarse.
- **Precondición**: Cómo se encuentra la situación previa realización del caso de uso.
- **Postcondición:** Una vez realizado el caso de uso, el estado de la situación o el sistema tras un desarrollo normal de éste.
- **Desarrollo normal**: Detalle de la interacción entre el actor del caso de uso y el sistema mediante una serie de acciones.
- **Desarrollo alternativo**: Puntos en los que puede darse una alternativa, cuando las acciones no son las del desarrollo normal.

Se explican, a continuación, una serie de casos de uso que demostrarán el comportamiento del sistema en relación al proceso de DevOps diseñado e implementado en este proyecto.

Caso de uso	***CU-01 – Control de versiones del código***	
Actores	Software de control de versiones	
Propósito	Gestionar las modificaciones que se realicen sobre un producto, en este caso, el código que el desarrollador esté generando.	
Visión general	El fin de disponer de un software de control de versiones, es tener una herramienta de almacenamiento del código sobre el que llevar un registro histórico de los cambios que se realizan sobre él.	
Tipo	Primario	
Precondición	Iniciar el sistema Crear un repositorio dentro de la herramienta con control de versiones	
Postcondición	Se almacena y modifica el código, y se mantiene un registro histórico de estas modificaciones. Se crean ramas con diferentes versiones de un mismo código fuente.	
Desarrollo normal	**ACTOR**	**SISTEMA**
	Modifica el código	Almacena el código. Registra cada modificación. Permite consultar las versiones antiguas del código modificado. Permite crear ramas en un mismo repositorio
Desarrollo alternativo	**ACTOR**	**SISTEMA**
	No aplica	No aplica

Tabla 23 - CU-01 - Control de versiones del código

Caso de uso	***CU-02 – Automatizar el análisis del código y la compilación***	
Actores	Servidor de automatización	
Propósito	Ejecutar análisis sobre el código automáticamente cuando el desarrollador actualice el código fuente en el repositorio remoto accesible mediante GIT, y su posterior compilación.	
Visión general	Contar con una herramienta que lance automáticamente los análisis estáticos sobre el código fuente cuando el desarrollador interactúe con el código que se encuentra en el repositorio remoto, así como otras tareas que se configuren, como la compilación del código.	
Tipo	Primario	
Precondición	Iniciar el sistema Crear un repositorio de código accesible mediante GIT Vincular el repositorio de código con el servidor de automatización Configurar un proyecto en la herramienta de análisis de código Parametrizar el servidor de automatización para vincular el repositorio de código, y la herramienta de análisis Actualizar el código en el repositorio	
Postcondición	Se ejecuta el análisis y se compila el código	
Desarrollo normal	**ACTOR**	**SISTEMA**
	Modificar el código en el repositorio remoto.	Ejecuta el análisis. Compila el código.
Desarrollo alternativo	**ACTOR**	**SISTEMA**
	Realiza una acción sobre el repositorio que no implica análisis de código.	No ejecuta nada.

Tabla 24 - CU-02 - Automatizar el análisis del código y la compilación

Caso de uso	***CU-03 – Análisis de calidad y seguridad del código***	
Actores	Plataforma de análisis estático de código fuente	
Propósito	Contar con una plataforma que analice las modificaciones en el código y proporcione información que mejore la calidad y seguridad del código desarrollado.	
Visión general	Disponer de las métricas de calidad generadas tras un análisis sobre el código fuente del repositorio. Disponer de información acerca de las soluciones posibles a diferentes cuestiones encontradas tras el análisis.	
Tipo	Primario	
Precondición	Iniciar el sistema Modificar el código en el repositorio remoto	
Postcondición	Se realiza el análisis y se generan métricas de calidad y seguridad del código Se muestran en un panel de control los resultados del análisis Se muestran soluciones a problemas encontrados mediante el análisis, si los encuentra	
Desarrollo normal	**ACTOR**	**SISTEMA**
	Actualizar el código en el repositorio remoto.	Analiza el código almacenado en el repositorio. Genera métricas de calidad y seguridad del código actualizado Muestra soluciones a problemas en el código
Desarrollo alternativo	**ACTOR**	**SISTEMA**
	No aplica	No aplica

Tabla 25 - CU-03 - Análisis de calidad y seguridad del código

3.4. Matrices de consistencia y trazabilidad

En este apartado se incluyen las matrices donde se reflejan la consistencia entre los requisitos del proceso, así como los posibles conflictos o redundancia entre ellos. A continuación de esta, la matriz de trazabilidad entre requisitos del proceso DevOps y casos de uso.

3.4.1. Matriz de consistencia entre requisitos

En la siguiente tabla se reflejan los conflictos, redundancias o acoplamientos que pudieran darse entre los requisitos del proceso DevOps que se definieron en el apartado 3.2.3 de este documento. No se encuentran ni conflictos, ni redundancias, pero sí acoplamientos. Estos se muestran sobre la tabla con el símbolo (+). En caso de existir algún conflicto, se reflejan con el símbolo (o), y las redundancias con (x).

Requisito	*RDB-01*	*RDB -02*	*RDIC-01*	*RDIC-02*	*RDIC-03*	*RDFC-01*	*RDFC-02*	*RDFC-03*	*RDFC-04*	*RDFC-05*
RDB-01		+	+							
RDB -02	+			+						
RDIC-01	+									
RDIC-02										
RDIC-03							+			
RDFC-01										
RDFC-02		+			+					
RDFC-03										+
RDFC-04										
RDFC-05								+		

Tabla 26 - Matriz de consistencia entre requisitos del proceso DevOps

Que se produzcan acoplamientos entre algunos requisitos se deben a que, entre ellos, están relacionados con la definición de la misma funcionalidad del proceso, sin ser redundantes. En este caso, algunos se acoplan por su relación con el control de versiones, y el otro grupo en el que se dan acoplamientos es el de los relacionados con el análisis del código.

3.4.2. Matriz de trazabilidad entre requisitos del proceso DevOps y casos de uso

Se refleja en la siguiente tabla la coherencia de los requisitos del proceso DevOps definidos, con los casos de uso identificados. Se marca la relación con el símbolo (+) cuando corresponde.

	RDB-01	*RDB -02*	*RDIC-01*	*RDIC-02*	*RDIC-03*	*RDFC-01*	*RDFC-02*	*RDFC-03*	*RDFC-04*	*RDFC-05*
CU-01	+	+								
CU-02			+	+	+					
CU-03						+	+	+	+	+

Tabla 27 - Matriz de trazabilidad requisitos DevOps - casos de uso

Puede identificarse que los casos de uso explicados 3.3.2 cubren las fases del proceso DevOps que abarca este proyecto, y que cubren los requisitos que se han definido para cada una de esas fases.

4. DISEÑO DEL SISTEMA

Una vez analizadas las capacidades y restricciones del sistema, en este capítulo se detallan las decisiones que se toman para proveer una solución al problema planteado en este proyecto. Incluye, por lo tanto, una explicación de las tecnologías que se escogen para crear la mencionada solución.

4.1. Arquitectura

En este apartado, se reflejan diferentes vistas de la arquitectura del sistema. Los diagramas que se muestran a continuación, así como los de apartados anteriores, se realizan con la herramienta diagrams.net que ofrece como extensión Google Drive, de forma gratuita. Se elige por la facilidad de uso, y su integración con Google Drive.

Para definir las vistas (lógica, de proceso, y, de desarrollo) de la arquitectura del software, está basado en el modelo de "4+1", de Philippe Kruchten, del artículo publicado en IEEE Software 12(6) (*Philippe Kruchten, 1995*).

Actualmente, existen diversos arquetipos arquitectónicos de software. Los principales son los siguientes:

- **Modelo monolítico**: Todas las funcionalidades acopladas en una misma aplicación. Su ventaja es que suele resultar más sencillo de desarrollar en primera instancia, pero aumenta en complejidad en función de lo que pretenda escalarse el sistema.
- **Arquitectura de tres capas, o monolítico 2.0**; Dividir en 3 capas monolíticas la capa de presentación, el servidor de aplicaciones, y la capa de base de datos. Más modular que el anterior, pero el escalado sigue siendo insuficiente.
- **Modelo modular, o, Micro servicios**. Dividir en unidades pequeñas e independientes el sistema, haciendo de cada unidad una aplicación en sí misma. Cada una de estas unidades es simple, su implementación y escalado es independiente de las otras. Acarrea otras desventajas, como puede ser que el número de unidades integradas sea muy numeroso. Otra desventaja es que requiere mayor administración de dependencias. Pero aporta mucha más flexibilidad que en los dos anteriores casos. (*Team, C., 2020*)

El sistema diseñado en este proyecto está basado en una arquitectura de micro servicios.

4.1.1. Vista lógica

En esta vista se usa un método de diseño orientado a objetos, para describir los objetos que componen el modelo. Esta vista declara principalmente la relación entre componentes del sistema que satisfacen los requisitos del proceso de DevOps diseñado en este proyecto. Para la representación, solo se tienen en cuenta los objetos relevantes para la arquitectura.

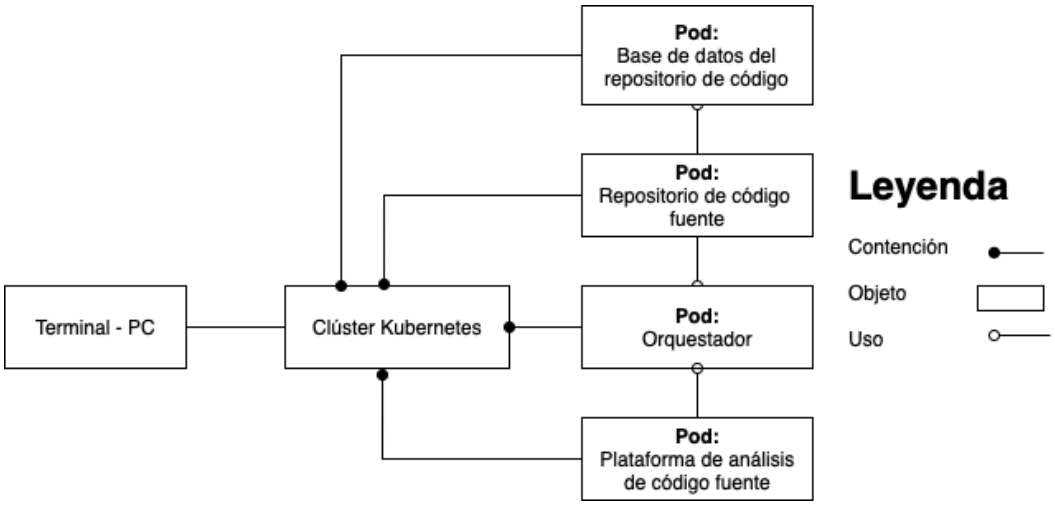


Ilustración 8 - Vista lógica o conceptual

Explicación de la relación de los componentes en el diagrama:

- Terminal – PC:
  - El equipo en el que está instalado el sistema
  - En él se despliega el contenedor donde están instaladas el resto de aplicaciones.
- Clúster de Kubernetes:
  - Sobre él se instalan las diferentes aplicaciones/pods (*vainas*) que conforman el sistema de Integración Continua.
  - Pods: es el objeto más simple que usa Kubernetes, que contiene al menos un contenedor de software. Contiene los de base de datos, repositorio de código fuente, orquestador, y la plataforma de análisis. (BENITEZ, A., 2018)
- Pod de base de datos:
  - Pod asociado a un volumen, en el que se persiste la información del repositorio de código. Por tanto, de él hace uso el repositorio de código.
- Pod de repositorio de código:
  - Despliega la aplicación del repositorio de código fuente.
  - Integrado con el orquestador, para que se puedan ejecutar las automatizaciones.

- Pod de orquestador:
    - Despliega la aplicación con la plataforma de automatización.
- Pod de plataforma de análisis:
    - Despliega la aplicación de la plataforma de análisis estático del código.
    - Integrado con el orquestador, para ejecutar la automatización de los análisis.

4.1.2. Vista de proceso

También conocida como vista de ejecución. En ella, se describen de forma gráfica las fases por las que puede pasar el sistema, en función del evento que lo dispare.

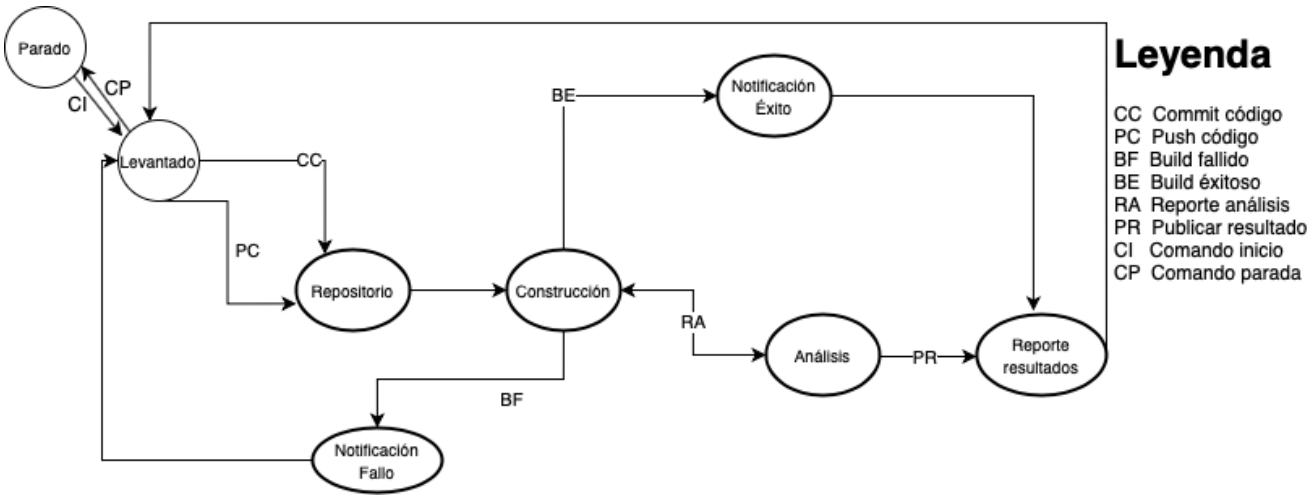


Ilustración 9 - Vista de proceso, o ejecución

Cada óvalo es una fase a la que se puede transitar, tal y como es diseñado el sistema. Pueden distinguirse las tres aplicaciones principales que componen el sistema, siendo el repositorio de código fuente, la fase de construcción, ejecutada por la aplicación que actúa de orquestador, y el análisis y reporte, que es ejecutado por la plataforma de análisis del código fuente.

Es posible observar que pueden realizarse dos acciones para interactuar con el repositorio de código, siendo estas hacer un push del código o un commit del código.

Se observa que el evento que se produce entre la construcción y el análisis es bidireccional. Esto es así porque desde la fase de construcción se ejecuta el análisis, y puede interrumpirse la construcción en función del resultado de dicho análisis.

Tanto desde la notificación de un fallo en la construcción (learnitguide net, 2018), como desde la publicación del resultado de los análisis, el sistema pasará al estado Levantado, a la espera de interacción por parte del usuario.

4.1.3. Vista de desarrollo

También conocida como vista de implementación, está enfocada a la organización de los módulos que componen el sistema de cara al desarrollo de éste. Se puede descomponer en módulos o capas, mostrando claramente las interfaces que cada componente ofrece hacia otros para su integración.

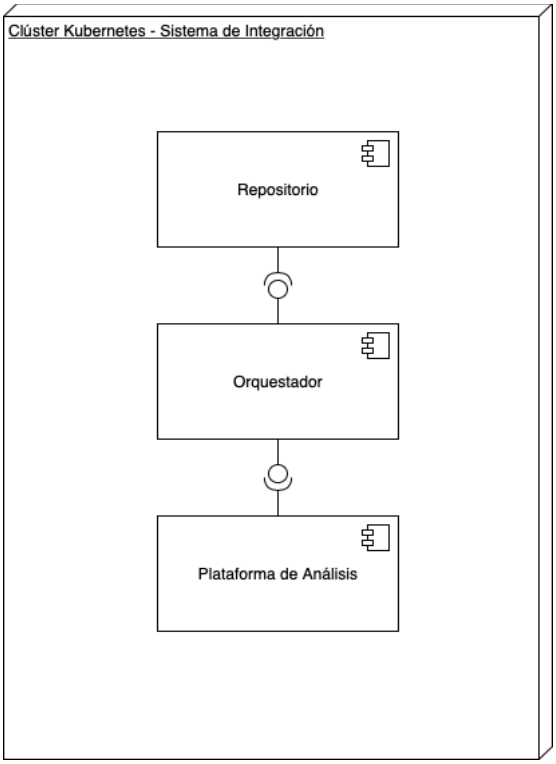


Ilustración 10 - Vista de desarrollo, o implementación

En el diagrama anterior, se muestran los módulos del sistema que están instalados en el clúster de Kubernetes y que forman nuestro sistema de integración continua.

El usuario desarrollador dispone de un repositorio de código con software de control de versiones, un software orquestador que ejecuta las acciones automatizadas, y una plataforma de análisis que realiza los análisis de calidad y seguridad, y a la que se puede acceder para consultar los resultados de estos análisis.

Con estas tres aplicaciones, se diferencian tres componentes dentro del sistema. Un componente es un elemento modular que puede sustituirse en su propio contexto. Sus elementos internos no se muestran, pero tiene una o varias interfaces propias definidas, a través de las cuales se puede hacer uso de sus capacidades.

- El componente de Repositorio, es el encargado de almacenar las modificaciones que vayan realizándose sobre el código fuente que el desarrollador necesite. Se comunica con el componente Orquestador mediante una interfaz que éste le ofrece, permitiéndole recoger información sobre los eventos realizados en el Repositorio, para poder lanzar tareas.
- El componente Orquestador, es el encargado de realizar acciones de forma automática en función del evento llevado a cabo sobre el repositorio de código, a través de los llamados Pipelines. Se comunica también con el componente Plataforma de Análisis, a través de una interfaz por la que tanto envía como recibe información.
- El componente Plataforma de Análisis, encargado de escanear el código actual del repositorio de código y muestra en su panel de control los resultados de éste. Los análisis se lanzan mediante acciones del componente Orquestador, y la Plataforma envía también información al orquestador para que lance otras acciones en función del resultado de los análisis.

4.2. Infraestructura tecnológica: Herramientas utilizadas

Para la composición del sistema, se han escogido las siguientes tecnologías:

- **Minikube**: Es una versión reducida y local de Kubernetes, que haciendo uso de un hipervisor levanta un clúster local de un solo nodo, haciendo que actúe de maestro y esclavos al mismo tiempo. Requiere menos recursos, y para el sistema que se ha elaborado es más que suficiente. Su instalación es sencilla, y existe mucha documentación por lo que su configuración resulta simple y rápida. Puede manejarse por comandos, como se enseña en el apartado de implementación, aunque también puede usarse mediante un panel de control que viene instalado por defecto, si fuera necesario.

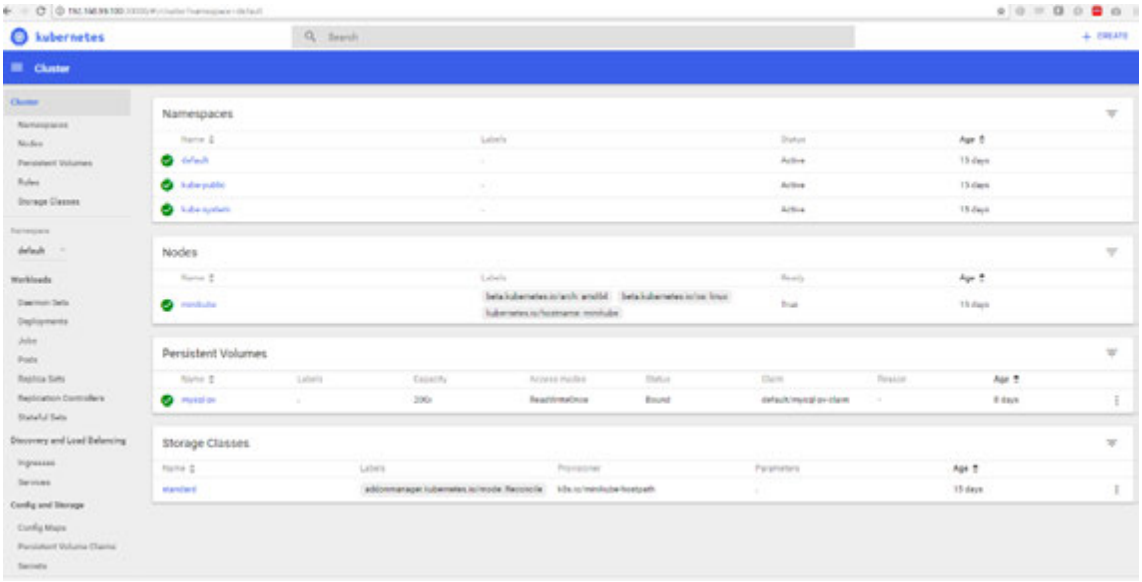


Ilustración 11 - Panel de control de Minikube

- **Gitea**: Es una herramienta de código abierto que permite alojar tus proyectos haciendo uso de un sistema de control de versiones del desarrollo utilizando Git. Se escoge Gitea, antes que otras más conocidas como GitLab o GitHub, por conocimiento previo y porque se que cuenta con una comunidad muy amplia de usuarios, por lo que existe mucha documentación para poder instalarlo en un sistema local e integrarlo con otras aplicaciones.

- **Jenkins**: Es un servidor de automatización, o de integración continua, de código abierto y gratuito. Probablemente el servidor de automatización más utilizado actualmente, por lo que existe mucha documentación en la que apoyarse en el momento de la instalación. Además de por la cantidad de documentación, Jenkins es el orquestador que se utiliza a nivel empresarial actualmente, por lo que es interesante conocer cómo es su funcionamiento. La diversidad de extensiones que pueden instalarse para poder integrar el resto de herramientas que conforman el sistema diseñado en este trabajo de fin de grado (TFG), también resulta determinante para elegir Jenkins como orquestador.

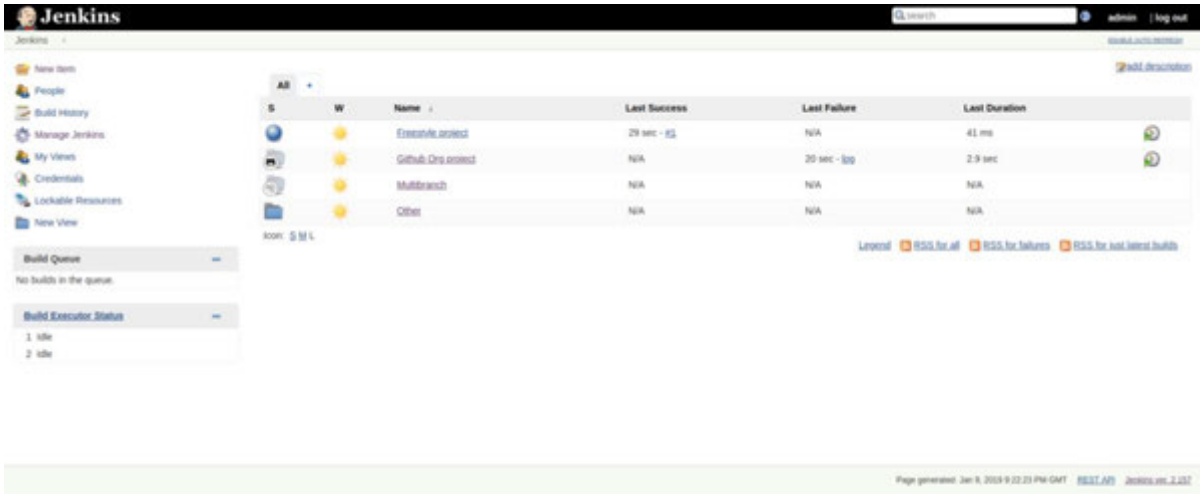


Ilustración 12 - Pantalla de inicio de Jenkins

- **SonarQube:** Es una plataforma de análisis de código estático. Analiza tanto la calidad como las posibles vulnerabilidades que pueda haber en nuestros desarrollos. Indica duplicidades, bugs, código muerto, qué porcentaje de nuestro código está cubierto por tests unitarios. En definitiva, es una herramienta muy útil para una constante mejora del código fuente. Al igual que ocurre con Jenkins, Sonar es la herramienta que se consulta en el día a día, por lo que interesa conocerla más en profundidad.

- **VirtualBox**: Se trata de un virtualizador, o hipervisor, desarrollado por Oracle Corporation. Es el hipervisor que se utilizará para desplegar la máquina virtual con Minikube sobre él.

5. IMPLEMENTACIÓN Y PRUEBAS

En este apartado de la documentación se especifica cuáles son los pasos seguidos para la creación del entorno de integración continua. A modo de guía, se indica paso a paso cómo se instalan las aplicaciones y se integran entre ellas.

5.1. Instalación de Minikube

Antes de instalar Minikube (Team, K., 2020), es necesario llevar a cabo una cierta preparación del equipo.

5.1.1. Pasos previos: Virtualización

En primer lugar, es necesario comprobar que en el equipo en el que se instala el entorno la virtualización está habilitada, en la BIOS del ordenador.

Como se instala sobre un ordenador cuyo sistema operativo es Linux, comprobar si está habilitada en BIOS se hace mediante un comando en consola:

```
egrep --color 'vmx|svm' /proc/cpuinfo
```

Si no estuviera habilitada, no se obtendría ningún tipo de salida por consola. En este caso, se obtiene la siguiente salida:

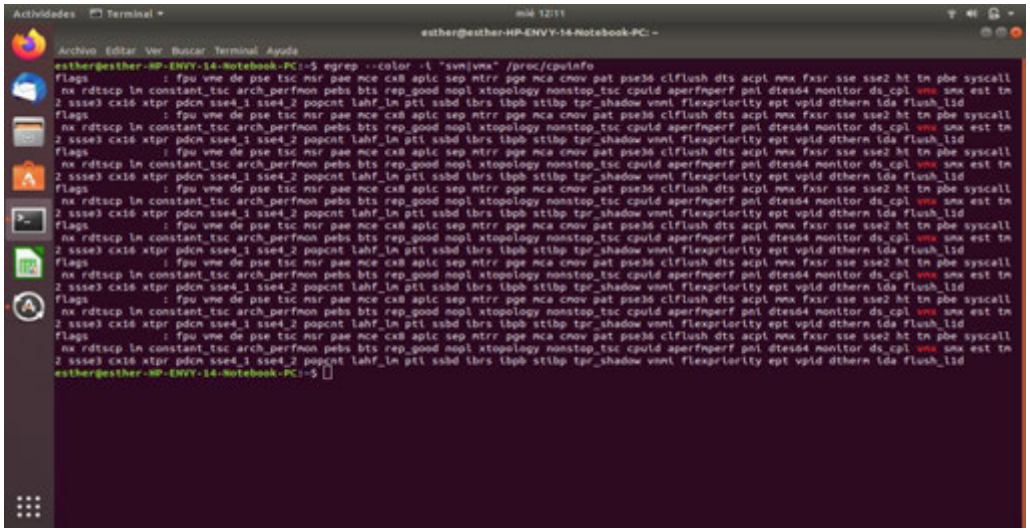


Ilustración 13 - Salida consola - Virtualización habilitada

Una vez comprobado esto, y habilitarla en caso de no estarlo, es necesario instalar un hipervisor si no se tiene ya instalado. En este caso, se utiliza VirtualBox porque ya está instalado en el equipo,

y porque es soportado tanto en Linux, Windows, y MacOS, satisfaciendo los requisitos RSP-01 y RSP-02.

5.1.2. Pasos previos: Kubectl

Es necesario instalar Kubectl, que es la herramienta de línea de comandos de Kubernetes (Kubernetes io, 2020). Eso es necesario para poder desplegar y gestionar aplicaciones en Kubernetes. Como se indica en la propia documentación de Minikube (kubernetes.io, 2020), puede instalarse de diversas maneras.

La forma más sencilla es utilizar Snap. Si se utiliza Ubuntu, a partir de la versión 16.04, se introduce un gestor de paquetes llamado Snap, que facilita la instalación de aplicaciones en Linux.

Para instalar Kubectl no es necesario más que utilizar el siguiente comando:

```
sudo snap install kubectl –classic
```

Se puede comprobar que se instala correctamente y consultar la versión instalada con el comando:

```
kubectl version
```

Con lo que imprime la siguiente salida por consola:

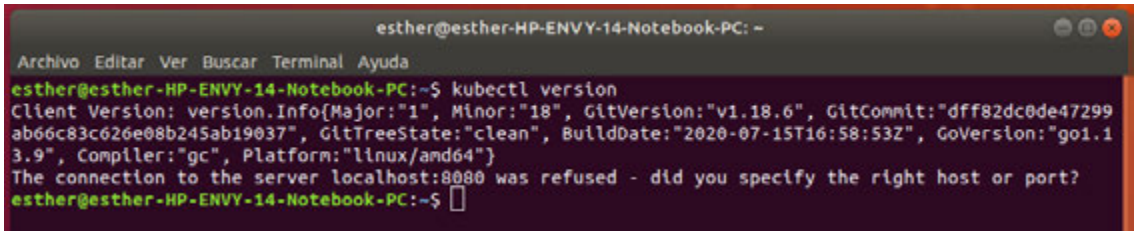


Ilustración 14 - Salida consola - Versión kubectl

5.1.3. Minikube

Una vez se tiene instalado lo anterior, se ejecuta el siguiente comando para descargar Minikube en el ordenador e instalarlo:

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Es posible comprobar que la instalación es correcta, inician el clúster con el comando:

```
minikube start
```

Pudiendo ver en la consola algo similar a lo siguiente:

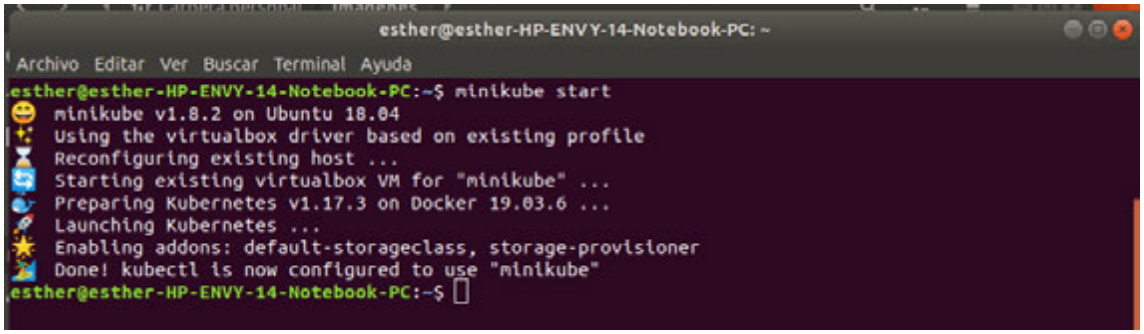


Ilustración 15 - Salida consola - Iniciar Minikube

Para desplegar un clúster indicando parámetros de configuración como capacidad de disco, el número de CPUs, o capacidad de memoria RAM, se realiza mediante el siguiente comando:

```
minikube start --cpus=4 --memory=12288 --disk-size=71680
```

En este caso, se indican 4 cpus, una capacidad de 12 gigabytes (GB) de memoria RAM, referido en megabytes, y una capacidad del disco de 70GB de almacenamiento.

Se configuran estas capacidades para asegurar que se dispone de suficientes recursos para probar diversas aplicaciones durante el desarrollo del proyecto.

5.2. Despliegue de aplicaciones

Una vez instalado e iniciado el clúster en Minikube, se procede a instalar las aplicaciones que van a formar el sistema.

5.2.1. Helm

Para desplegar las diversas aplicaciones, se utiliza una herramienta llamada Helm, que se usa para el despliegue de componentes de Kubernetes. Estos paquetes se denominan Charts. Explicar la definición de chart puede resultar complejo. Si Helm es un gestor de paquetes de Kubernetes, un chart es lo que sirve para empaquetar y desplegar una aplicación, o un componente de una aplicación más grande, que puede contener varios objetos de Kubernetes que son desplegados todos de una misma vez.

Para instalar Helm, se hace uso del siguiente comando, usando snaps de nuevo:

```
sudo snap install helm –classic
```

En pos de instalar diversas aplicaciones, se pueden consultar todos los Charts oficiales que ofrece Helm en su repositorio de GitHub: https://github.com/helm/charts

Para poder hacer uso de los charts, se tienen que añadir a Helm los repositorios que los contienen (Authors, 2020).

En nuestro caso, los repositorios de los que se ha hecho uso son los siguientes:

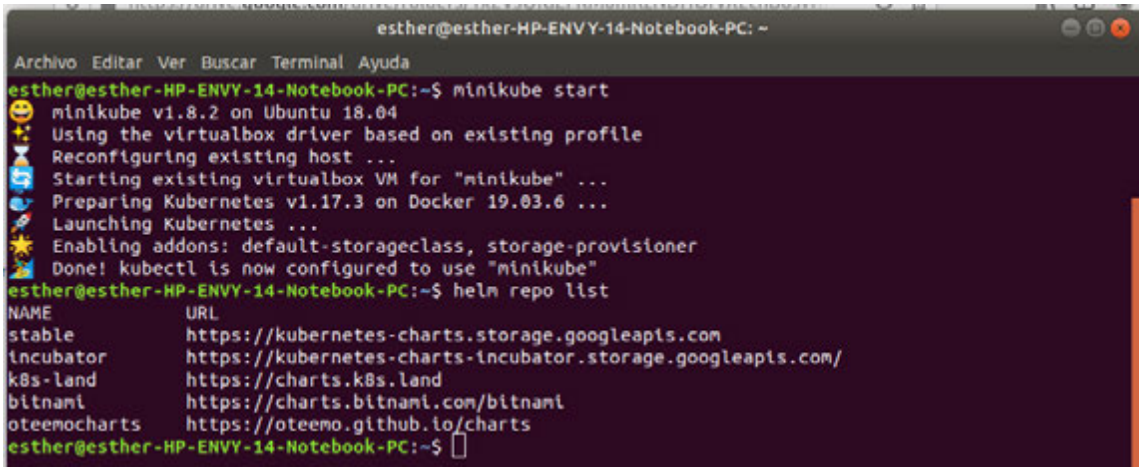


Ilustración 16 - Salida consola - Repositorios de Helm Charts

Para añadir cada uno de ellos, hay que hacer uso del siguiente comando:

```
helm repo add [nombre] [URL completa] [flags]
```

A continuación, se explica cómo se hace uso de ellos para instalar las aplicaciones en el entorno que se está configurando para este proyecto.

5.2.2. Instalación Gitea

Como se indica anteriormente, como herramienta de control de código fuente se hace uso de Gitea.

Lo primero que se lleva a cabo es añadir a los repositorios de Helm, del entorno local, el repositorio que contiene el chart de Gitea:

```
helm repo add k8s-land https://charts.k8s.land
```

Se puede instalar Gitea con los valores predeterminados, o se pueden editar los valores de su archivo de configuración antes de instalarlo para adaptarlo mejor a las necesidades del proyecto.

En este caso, se edita el archivo de la siguiente manera:

- En primer lugar, se descarga al equipo el archivo gitea-values.yaml para poder editarlo, con el siguiente comando:

```
helm show values k8s-land/gitea > gitea-values.yaml
```

- Dentro del fichero que se descarga, interesan dos fragmentos en especial:

```
mariadb:
  ## Whether to deploy a mariadb server to satisfy the applications database requirements. To use an external database set this to false and configure the externalDatabase parameters
    enabled: true
```

Figura 1 - Habilitar mariadb para gitea

- En esta parte del archivo se indica que, cuando se instale Gitea, se desplegará también una base de datos MariaDB para la aplicación de SCM (Source code management), Gitea.

```
service:
  http:
    serviceType: NodePort
    port: 3000
```

Figura 2 - Cambiar tipo de servicio para gitea

- Por defecto, el tipo de servicio viene como ClusterIP. En este proyecto, interesa cambiarlo a NodePort para que se pueda acceder desde el navegador.

Una vez cambiado esto, y comprobado que está habilitada la base de datos MariaDB, se puede desplegar Gitea haciendo uso de estos nuevos valores. Esto se hace con el comando:

```
helm install gitea k8s-land/gitea -f gitea-values.yaml
```

Una vez instalado, se pueden consultar los servicios que se han levantado en el clúster local con el comando:

```
kubectl get services
```

```
esther@esther-HP-ENVY-14-Notebook-PC:~/tfg$ kubectl get services
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
gitea-gitea-http        NodePort       10.109.242.143   <none>        3000:32334/TCP   83d
gitea-gitea-ssh         LoadBalancer   10.101.63.16     <pending>     22:30420/TCP     83d
gitea-mariadb           ClusterIP      10.100.157.54    <none>        3306/TCP         83d
```

Ilustración 17 - Salida consola - get services gitea

El que es relevante en este caso, para poder acceder a él desde el navegador es el servicio llamado gitea-gitea-http. Se pueden ver dos puertos diferentes para dicho servicio. El puerto 3000 será el puerto interno en el clúster. El puerto 32334 es el puerto de acceso externo. Se puede acceder al panel de control de Gitea desde el navegador mediante la ip de la máquina virtual, seguida del puerto 32334, que es con el que se acceder desde el navegador para este servicio. O bien, utilizando el comando 'minikube service [nombreDelServicio]':

```
minikube service gitea-gitea-http
```

Saliendo lo siguiente por consola, y abriéndose la aplicación en nuestro navegador:

```
esther@esther-HP-ENVY-14-Notebook-PC:~$ minikube service gitea-gitea-http
|-----------|------------------|-------------|---------------------------|
| NAMESPACE |       NAME       | TARGET PORT |            URL            |
|-----------|------------------|-------------|---------------------------|
| default   | gitea-gitea-http | http        | http://192.168.99.101:32334 |
|-----------|------------------|-------------|---------------------------|
🎉  Opening service default/gitea-gitea-http in default browser...
esther@esther-HP-ENVY-14-Notebook-PC:~$
```

Ilustración 18 - Salida consola - Service Gitea

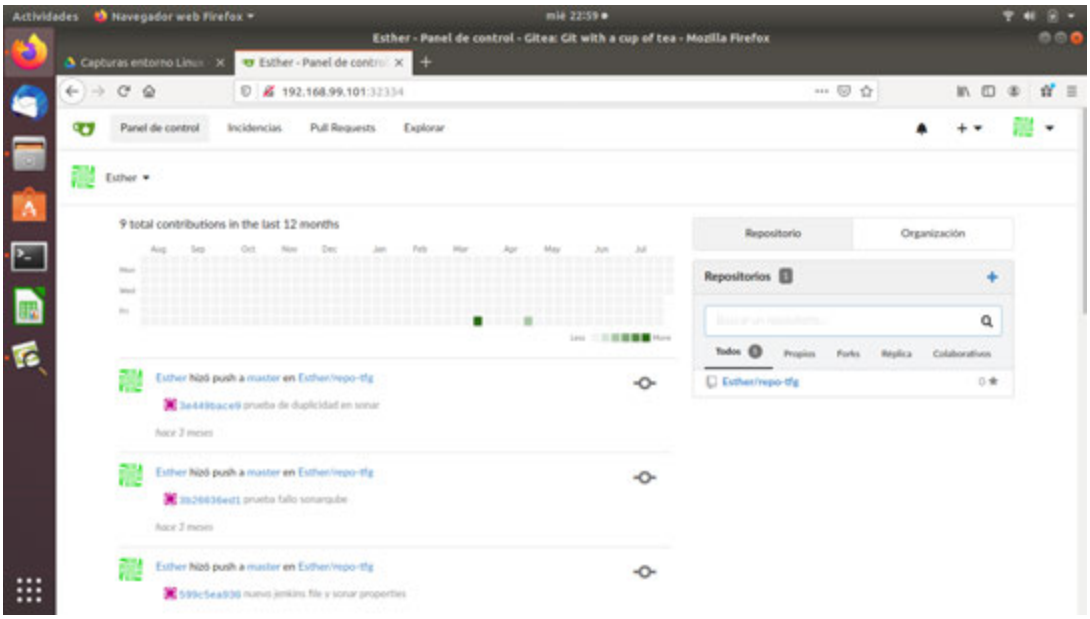


Ilustración 19 - Panel de control de Gitea en el navegador

5.2.3. Instalación de Jenkins

Como elemento orquestador, se procede a instalar, o desplegar, Jenkins. El proceso de despliegue es muy similar al seguido con el despliegue de Gitea.

Para poder modificar los valores por defecto del despliegue de Jenkins, se procede a descargar en el ordenador el jenkins-values.yaml de la aplicación y, una vez editado, se despliega con la nueva configuración.

Para poder descargar el archivo, se hace uso del comando:

```
helm show values stable/jenkins > jenkins-values.yaml
```

Así como en la instalación de Gitea se hace referencia al repositorio de Helm k8s-land, en este caso se puede ver que referencia al componente en el repositorio "stable".

Lo único que se altera en este fichero será lo siguiente:

```
# For minikube, set this to NodePort, elsewhere use LoadBalancer
# Use ClusterIP if your setup includes ingress controller
  serviceType: NodePort
```

Figura 3 - Tipo de servicio para Jenkins

Como se puede ver, en el propio archivo de configuración indica que, si se está utilizando minikube, hay que cambiarlo a NodePort. Por defecto viene el valor ClusterIP, pero en este caso se quiere acceder desde el navegador. Igual que ocurre con Gitea, y se apunta posteriormente con SonarQube, es necesario cambiar este valor por NodePort.

Una vez modificado el archivo, se procede al despliegue del componente con el archivo de configuración nuevo. El despliegue se realiza con el comando:

```
helm install jenkins stable/jenkins -f jenkins-values.yaml
```

Una vez realizado el despliegue, se pueden consultar los servicios actualmente levantados. Si todo ha salido correctamente, se reflejan en la consola de comandos, tanto los servicios previos (Gitea) como los levantados para Jenkins.

Y se listan los servicios levantados, en la salida por consola:

```
esther@esther-HP-ENVY-14-Notebook-PC:~$ kubectl get services
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
gitea-gitea-http        NodePort       10.109.242.143   <none>        3000:32334/TCP   130d
gitea-gitea-ssh         LoadBalancer   10.101.63.16     <pending>     22:30420/TCP     130d
gitea-mariadb           ClusterIP      10.100.157.54    <none>        3306/TCP         130d
jenkins                 NodePort       10.102.145.20    <none>        8080:32762/TCP   130d
jenkins-agent           ClusterIP      10.107.184.127   <none>        50000/TCP        130d
kubernetes              ClusterIP      10.96.0.1        <none>        443/TCP          130d
```

Ilustración 20 - Salida consola - Servicios levantados

Una vez comprobado esto, se pueden realizar los mismos pasos que se siguen al instalar Gitea, para poder acceder al servicio, en el navegador. En este caso el comando es:

```
minikube service jenkins
```

```
esther@esther-HP-ENVY-14-Notebook-PC:~$ minikube service jenkins
|-----------|---------|-------------|-----------------------------|
| NAMESPACE |  NAME   | TARGET PORT |             URL             |
|-----------|---------|-------------|-----------------------------|
| default   | jenkins | http        | http://192.168.99.101:32762 |
|-----------|---------|-------------|-----------------------------|
🎉  Opening service default/jenkins in default browser...
```

Ilustración 21 – Apertura de Jenkins en el navegador

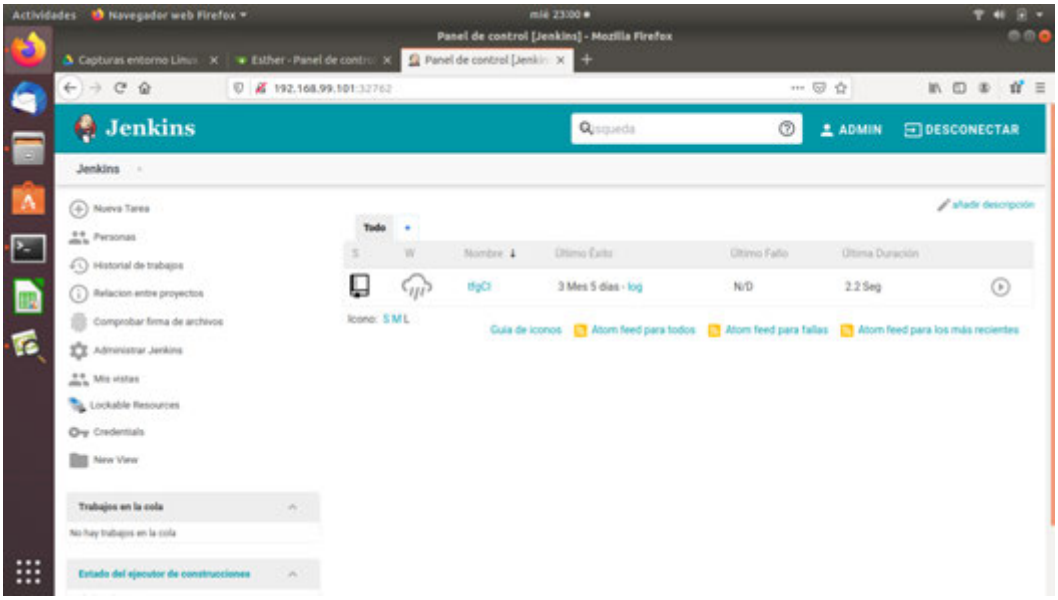


Ilustración 22 - Panel de control de Jenkins

En este caso, se aplica un cambio de aspecto a Jenkins que se puede hacer fácilmente desde el panel de configuración del propio Jenkins. Es sólo un cambio estético, por lo que no influye en la utilización de la herramienta.

En el apartado de integración de las herramientas se muestra cómo crear las tareas de automatización que se ejecutan desde Jenkins.

5.2.4. Instalación de SonarQube

Para la instalación de SonarQube, la herramienta de análisis elegida, se parte de una situación similar a la de la instalación de Gitea, en cuanto al almacenamiento de los datos de la aplicación. En el caso de Gitea, el archivo de configuración facilita la opción de habilitar MariaDB a la vez que se desplegaba Gitea por primera vez. En el caso de SonarQube, se opta por instalar una base de datos dedicada, y se utiliza PostgreSQL.

En primer lugar, se debe incluir en los repositorios de Helm el repositorio que contiene el chart de PostgreSQL actualizado. Se puntualiza esto, dado que en el repositorio stable se encuentra también un chart de PostgreSQL, pero en la propia documentación se remite a otro distinto dado que el suyo está obsoleto. Para añadir el repositorio:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

Para poder desplegar esta aplicación, y que concuerde con la configuración que se indica para SonarQube, es necesario modificar de nuevo su archivo de configuraciones. Se descarga el archivo:

```
helm show values bitnami/postgresql > postresql-values.yaml
```

En él, se modifican los valores por defecto de usuario, contraseña y base de datos para que concuerde con la configuración de SonarQube.

```
postgresqlUsername: sonarUser
postgresqlPassword: sonarPass
postgresqlDatabase: sonarDB
```

Figura 4 - Parámetros de configuración de PostgreSQL

En este caso, no se cambia el tipo de servicio a NodePort, sino que permanece como ClusterIP. Esto es porque no es necesario acceder desde el navegador, sino que es suficiente que SonarQube se comunique con dicho servicio de manera interna en el clúster.

Una vez hecho esto, para el despliegue el procedimiento igual que en las dos anteriores aplicaciones, haciendo uso del comando:

```
helm install postgre bitnami/postgresql -f postresql-values.yaml
```

Ya instalada la base de datos, se puede instalar la herramienta de análisis de código estático del sistema, SonarQube.

En primer lugar, añadir el repositorio de charts que mantiene actualmente el chart de SonarQube:

```
helm repo add oteemocharts https://oteemo.github.io/charts
```

Una vez añadido el repositorio, se descarga el archivo de configuración:

```
helm show values oteemocharts/sonarqube > sonarqube-values.yaml
```

En este caso, los cambios que se han de realizar son, además de cambiar el tipo de servicio como se hizo con Jenkins o Gitea, comprobar que coinciden los parámetros de configuración de PostgreSQL y apuntar al servicio de PostgreSQL desplegado.

```
postgresql:
  # Enable to deploy the PostgreSQL chart
  enabled: false
  postgresServer: "postgre-postgresql.default.svc.cluster.local"
```

Figura 5 - Configuración de los valores de SonarQube

Donde "default" es el nombre del *workspace* (espacio de trabajo) en Minikube. Es necesario adaptarlo si se le da un nombre concreto al workspace en el momento en el que se hace la instalación de Minikube.

```
service:
  type: NodePort
```

Figura 6 - Tipo de servicio SonarQube

Una vez modificado, se procede igual que con las anteriores aplicaciones. Se despliega SonarQube apuntando al archivo de configuración modificado:

```
helm install sonarqube oteemocharts/sonarqube -f sonarqube-values.yaml
```

Una vez instalado SonarQube, se puede comprobar que todos los servicios están instalados correctamente, que están levantados, y terminan las instalaciones de las aplicaciones. Una vez comprobado, se pasa a integrar las aplicaciones para que el funcionamiento del sistema sea como el que se diseña en apartados anteriores de este documento.

Para comprobar cuáles son los servicios desplegados en nuestro clúster, el comando será el mismo que el utilizado anteriormente.

```
esther@esther-HP-ENVY-14-Notebook-PC:~/tfg$ kubectl get services
NAME                        TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
gitea-gitea-http            NodePort       10.109.242.143   <none>        3000:32334/TCP   83d
gitea-gitea-ssh             LoadBalancer   10.101.63.16     <pending>     22:30420/TCP     83d
gitea-mariadb               ClusterIP      10.100.157.54    <none>        3306/TCP         83d
jenkins                     NodePort       10.102.145.20    <none>        8080:32762/TCP   83d
jenkins-agent               ClusterIP      10.107.184.127   <none>        50000/TCP        83d
kubernetes                  ClusterIP      10.96.0.1        <none>        443/TCP          83d
postgre-postgresql          ClusterIP      10.108.153.244   <none>        5432/TCP         83d
postgre-postgresql-headless ClusterIP      None             <none>        5432/TCP         83d
sonarqube-sonarqube         NodePort       10.104.38.61     <none>        9000:31081/TCP   83d
esther@esther-HP-ENVY-14-Notebook-PC:~/tfg$
```

Ilustración 23 - Instalación completada

Como se realiza con Gitea o Jenkins, es posible abrir en el navegador SonarQube.

```
minikube service sonarqube-sonarqube
```

```
esther@esther-HP-ENVY-14-Notebook-PC:~$ minikube service sonarqube-sonarqube
|-----------|---------------------|-------------|---------------------------|
| NAMESPACE |        NAME         | TARGET PORT |            URL            |
|-----------|---------------------|-------------|---------------------------|
| default   | sonarqube-sonarqube |             | http://192.168.99.101:31081 |
|-----------|---------------------|-------------|---------------------------|
  Opening service default/sonarqube-sonarqube in default browser...
```

Ilustración 24 - Apertura SonarQube en el navegador

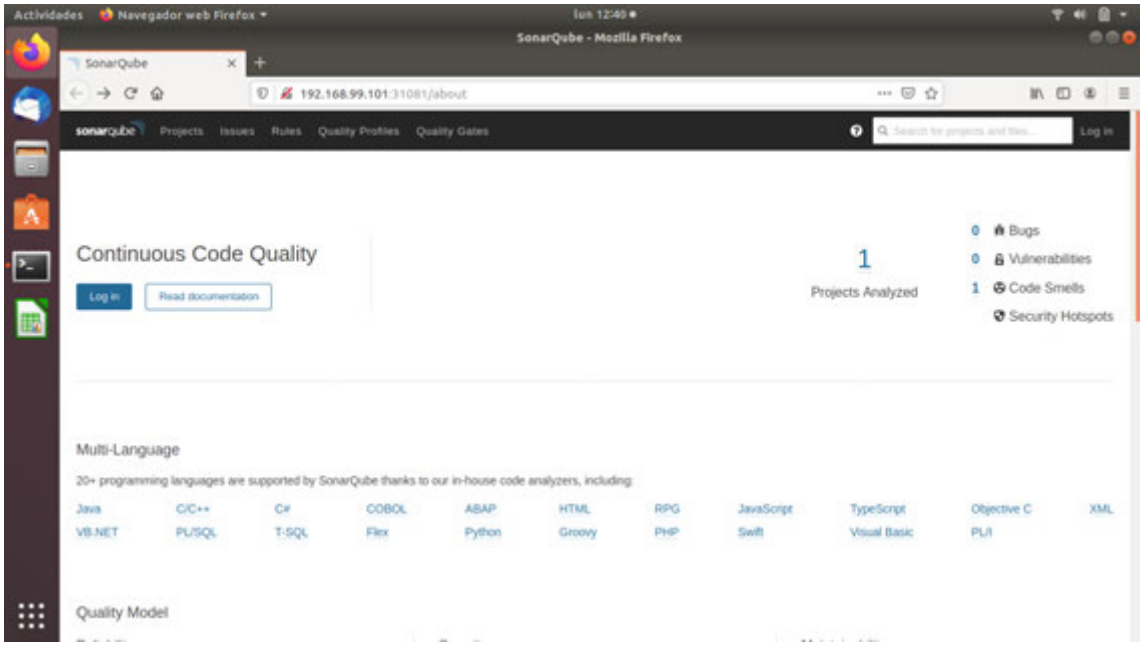


Ilustración 25 - Panel de control de SonarQube

5.2.5. Configuración de umbrales de calidad y generación de métricas de calidad

Una vez hemos instalado SonarQube, se pueden configurar los umbrales de calidad que aplicarán al proyecto que se genere. La creación de un proyecto en SonarQube se explica en el apartado 5.3.2 de esta documentación; Por defecto, la herramienta aplica un umbral por defecto para el lenguaje de programación que se configura para el proyecto (*SonarSource, 2020*).

Estos umbrales aplican reglas de calidad y seguridad cuando se realiza el análisis sobre el código alojado en el repositorio, y se generan unas métricas de calidad que son necesarias para determinar aspectos particulares del código, como su mantenibilidad, seguridad, reusabilidad, fiabilidad o portabilidad. Se pueden crear nuevos umbrales con reglas ya existentes, o bien crear reglas específicas para obtener más información acerca de la calidad del uso que se hace del proceso de DevOps, además de la calidad del propio código.

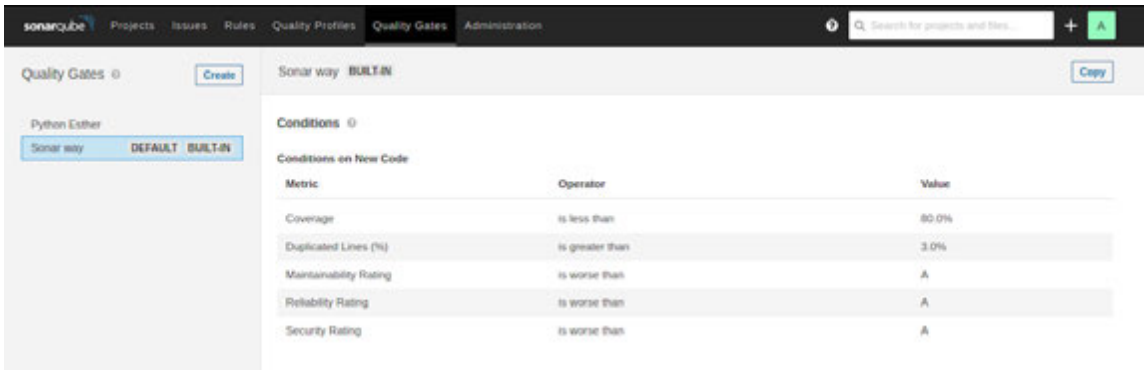


Ilustración 26 - Umbral de calidad por defecto en SonarQube

En la ilustración 26, se pueden ver los porcentajes que no pueden superarse en los 5 aspectos que se han mencionado en el párrafo anterior, para que se considere que es un código admisible.

Podemos consultar cuales son las reglas activas para este umbral de calidad, que posteriormente generarán las métricas de calidad que ayudarán a evaluar las propiedades del código analizado.

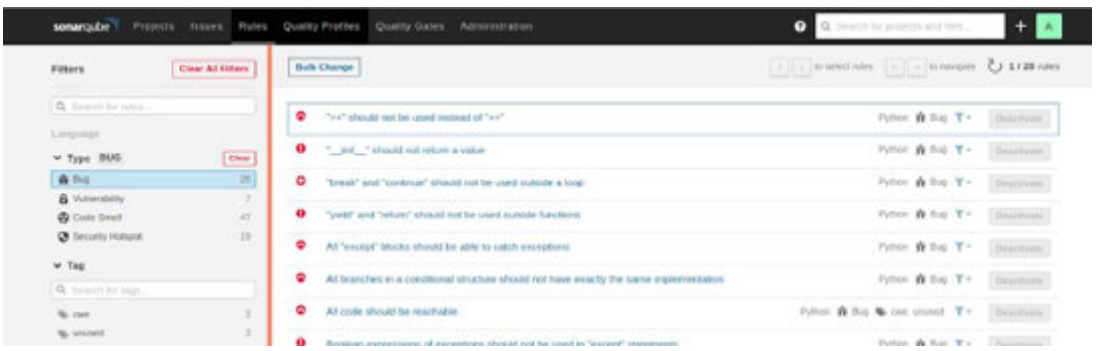


Ilustración 27 - Reglas de calidad activas en SonarQube

Para crear un nuevo umbral de calidad, en el apartado de "Quality profiles", como se puede ver en la ilustración 26, existe una funcionalidad "Create". En este caso, se crearía un umbral con las reglas disponibles para el lenguaje de código elegido. Una vez hecho esto, se asocia el nuevo umbral al proyecto de análisis creado, para que comiencen a aplicar las reglas que se hayan configurado.

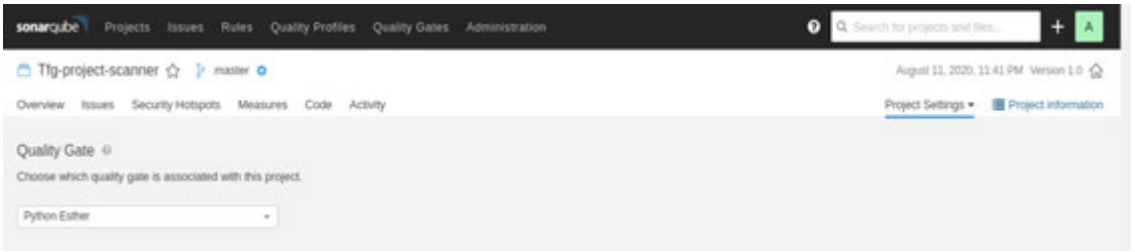


Ilustración 28 - Configurar nuevo umbral de calidad al proyecto

Para la creación de nuevas reglas, en el caso de Python, lenguaje de pruebas para este diseño del proceso DevOps, es necesario crear un complemento (plug-in) para SonarQube, e instalarlo directamente en la plataforma para poder utilizarlas.

5.3. Integración de las aplicaciones desplegadas

Una vez desplegados todos los componentes requeridos en el clúster, es necesario conectar los servicios entre ellos para que funcionen de forma automática.

Como se apunta en el apartado de diseño del sistema, es Jenkins la aplicación conectada con Gitea y con SonarQube, y no es necesario conectar Gitea y SonarQube directamente entre ellos.

Integrar Jenkins con otras aplicaciones no es complejo. Dispone de una biblioteca de complementos (*plug-ins*) amplia, donde es posible encontrar el complemento tanto para Gitea como para SonarQube.

5.3.1. Conexión Gitea – Jenkins

A continuación, se guía cómo se integra Gitea con Jenkins, para que se lance la ejecución del *pipeline* cuando se produce un cambio en el repositorio de código (Mike's Software Blog, 2019). Un pipeline de Jenkins es un conjunto de etapas, o instrucciones, que sigue el proceso ciclo de vida de un software desde que se incluye en el repositorio de código hasta que llega a los usuarios finales.

En este caso, el *pipeline* que se define no llega hasta una publicación para usuarios finales. Esto se explica posteriormente en esta documentación.

En primer lugar, cabe decir que la conexión entre ambas aplicaciones, además de mediante un complemento que se instala en Jenkins para Gitea, se hace a través de un mecanismo llamado Webhook.

Para descargar en Jenkins el complemento de Gitea, se accede a la ruta Administrar Jenkins > Administrar complementos, y en ella se busca Gitea Plugin, se instala y se reinicia Jenkins.

Una vez está instalado el plugin, se procede a conectar ambas aplicaciones.

En primer lugar, se añade a Jenkins las credenciales de acceso al repositorio de Gitea. Esto se realiza en el propio panel de control de Jenkins, apartado "Credentials"

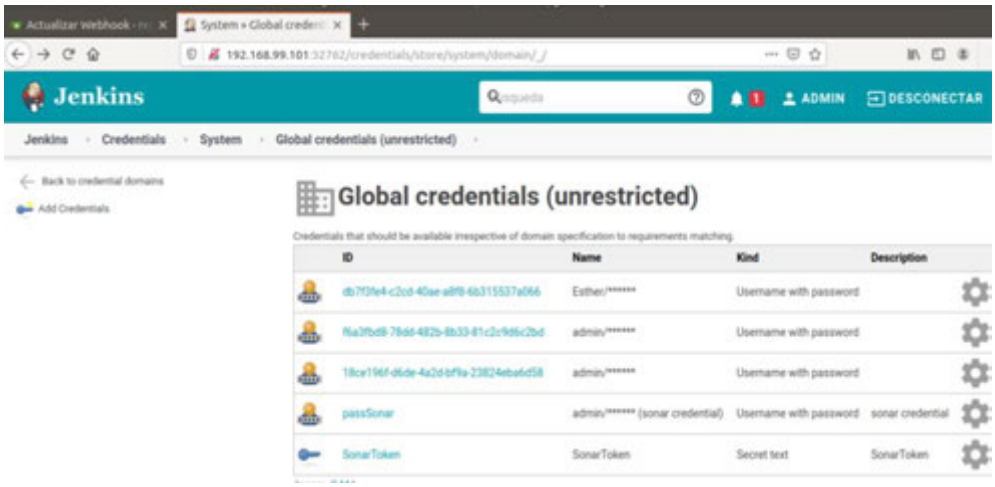


Ilustración 29 - Apartado de credenciales y tokens de Jenkins

Una vez añadidas las credenciales, se accede en la ruta Administrar Jenkins > Configuración del sistema, y buscar el punto "Gitea Servers", donde se apunta al servicio de Gitea del clúster local.

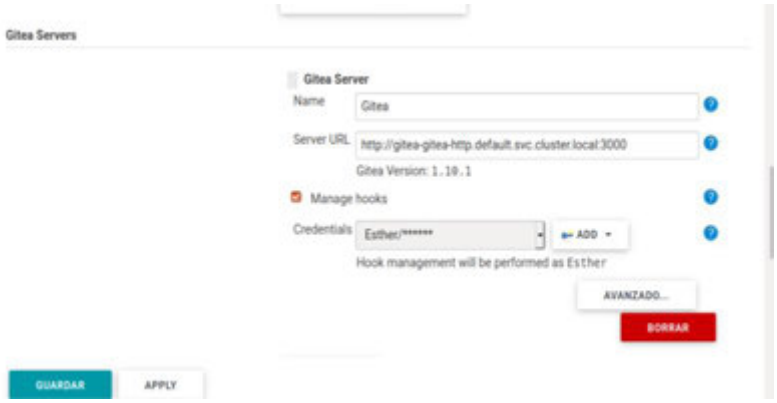


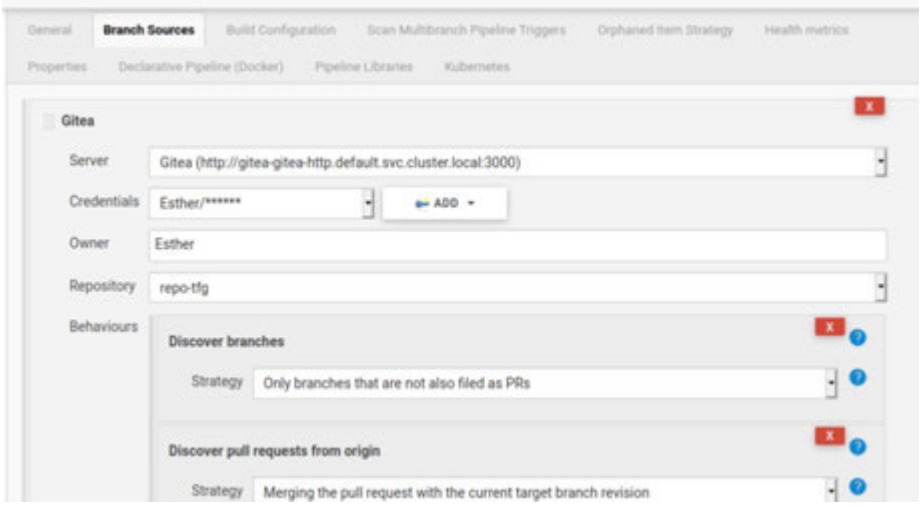
Ilustración 30 - Configuración Gitea Server

Se puede ver en la Ilustración 30 que, además de indicar las credenciales que ya se habían guardado previamente, precisa indicar la URL del servicio de Gitea dentro del clúster, y que el puerto de acceso es el puerto interno (esto puede corroborarse con lo que muestra la Ilustración 17).

Una vez hecho esto, se crea la tarea, o *Job*, que se ejecuta tras las interacciones con el repositorio de código. Para ello, se selecciona en el panel de control de Jenkins la opción "Nueva tarea". Es posible elegir entre diversas clases de *jobs*. En este caso, se elige la opción "*Multibranch Pipeline*" (Pipeline multirrama), porque se pueden ejecutar distintas tareas para el ciclo de vida del software, en función de la rama sobre la que se trabaje en el repositorio de código remoto.

En la configuración, además de darle un nombre al job, que se nombra en este caso **tfgCI**, se pueden administrar distintos aspectos como la fuente de las ramas de código, o las configuraciones de los *Build* (las construcciones). En las siguientes capturas, se puede distinguir cuáles son los parámetros que se configuran en cada caso.

Para configurar la fuente de las ramas de código se indica fuente de tipo Gitea, y en los apartados de servidor y credenciales se indican las mismas que se han configurado previamente en el Gitea Server. En el apartado "*Repository*" (repositorio) hay que indicar el nombre del repositorio que se ha creado en Gitea. En este caso, es "**repo-tfg**", pero puede ser cualquiera que se genere en el servidor. Hay que asegurarse de que coincide exactamente con el nombre que tiene el repositorio creado. En el apartado "*Owner*" (dueño) hay que indicar el usuario creador del repositorio, en este caso es "Esther". El resto de los apartados permanecen con los valores por defecto, ya que no influyen en este sistema.



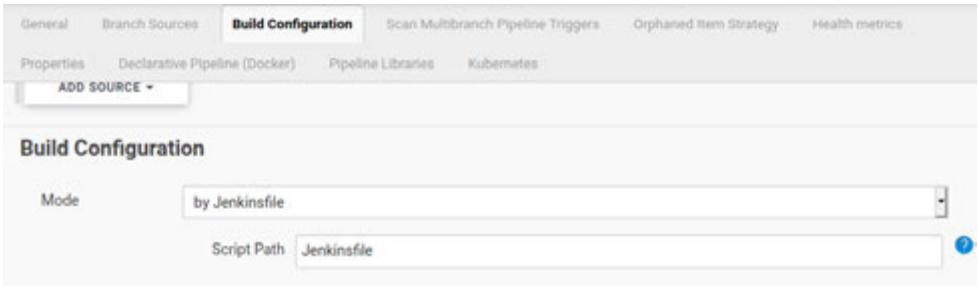


Ilustración 31 - Configuración de la tarea de Jenkins

La definición de las etapas que componen la fase de construcción se hace a partir de un fichero de texto llamado jenkinsfile. Este fichero se tiene subir al directorio principal del repositorio.

Un ejemplo, si se quisiera que cuando se hace un push del código a master se realice un después de la compilación del código, un análisis de calidad, pero no se quiere que pase tras un push sobre la rama develop, esto se indica en el fichero jenkinsfile. Este fichero no tiene por qué tener la misma configuración en todas las ramas. Se muestra, en el apartado de pruebas, cómo influye el contenido del fichero en las compilaciones.

Una vez que se define el *job*, se configura el webhook en el repositorio de Gitea.

Dentro del panel de control de Gitea, se selecciona el repositorio de código, y en el apartado de configuración, seleccionamos '*Webhooks'*. Una vez se pulsa sobre "Añadir Webhook", observamos los diversos tipos que pueden seleccionarse. En este caso, se utiliza del tipo Gogs (otro tipo de repositorio similar a Gitea), dado que en el momento en el que se estaba implementando el sistema, el webhook de tipo Gitea no funcionaba correctamente.

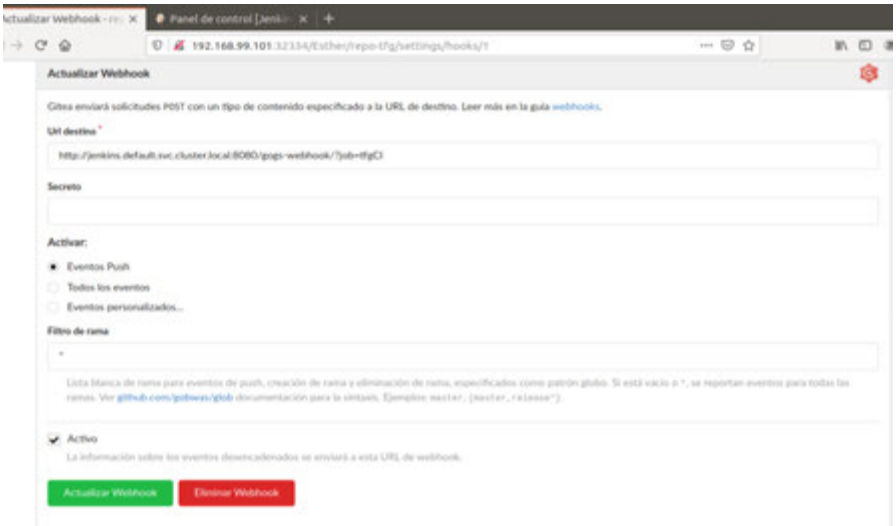


Ilustración 32 - Configuración Webhook Gitea-Jenkins

Como se puede ver en la Url de destino (en la Ilustración 32), apunta al jenkins instalado en el clúster local, con el puerto interno del servicio en el clúster. Lo siguiente "/gogs-webhook/" con lo que se indica que es un webhook de tipo gogs y como último parámetro de la url, se indica el nombre que le se ha dado al job en Jenkins, en este caso "?job=tfgCI". Activa las llamadas con los eventos push en el repositorio. Y se podría filtrar por rama, pero con "*" se indica que aplica para todas las ramas. Como puede verse en la Ilustración 32, se distingue que puede actuar el Webhook con todos los eventos, pero en este caso se deja indicado que escuche los eventos de tipo push.

5.3.2. Conexión SonarQube – Jenkins

Se detallan los pasos a seguir para integrar el análisis de código con SonarQube con el servicio de Jenkins del sistema.

En primer lugar, se instala en Jenkins el complemento para SonarQube. Se accede al administrador de complementos (*plug-ins*), y se busca SonarQube Scanner for Jenkins.

Abrir en el navegador el servicio de SonarQube, e iniciar sesión como usuario Administrador, siendo el usuario por defecto. Crear un proyecto para vincular con Jenkins, pulsando en el símbolo "+" a la izquierda del icono de la cuenta de usuario. Sólo pide otorgarle unos identificadores. Para poder configurar el plugin "SonarQube Server" como se hizo con el de Gitea en la configuración de sistema de Jenkins, es necesario crear un Token de autenticación para usarlo como credenciales para conectarlo con el orquestador, y poder conectar el proyecto recién creado en SonarQube.

Para generar este Token, acceder en SonarQube a *MyAccount* > *Sectury* > *Generate Token.*

Este token se añade en el apartado de "*Credentials"* de Jenkins para usarlo a posteriori.

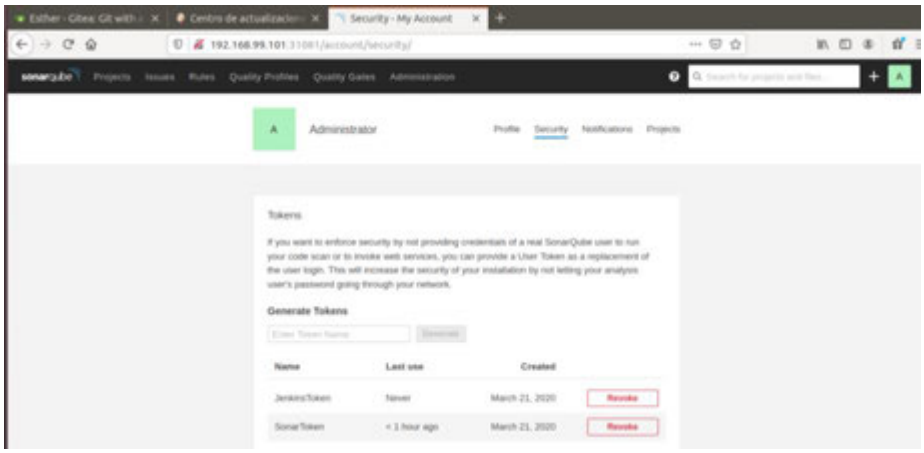


Ilustración 33 - Tokens de SonarQube

Dentro de SonarQube, crear un proyecto. En este caso, se nombra e indica la clave del proyecto como:

Tfg-project-scanner

Para asociar una rama a este proyecto se debe subir a al repositorio de código un archivo 'sonar-project.properties'. En este archivo se configura el nombre y clave del proyecto de sonar, y el lenguaje de programación del análisis. Siendo "py", el valor para Python.

```
# Metadata
sonar.projectKey=Tfg-project-scanner
sonar.projectName=Tfg-project-scanner
sonar.projectVersion=1.0

# Language
sonar.language=py
```

Figura 7 - Archivo de propiedades Sonar

A continuación, se parametriza en Configuración del Sistema, desde Jenkins, el servidor de SonarQube. La dirección URL del servidor se compone del nombre del servicio de SonarQube en el clúster local, el nombre del espacio de trabajo (default), seguido del puerto interno del servicio.

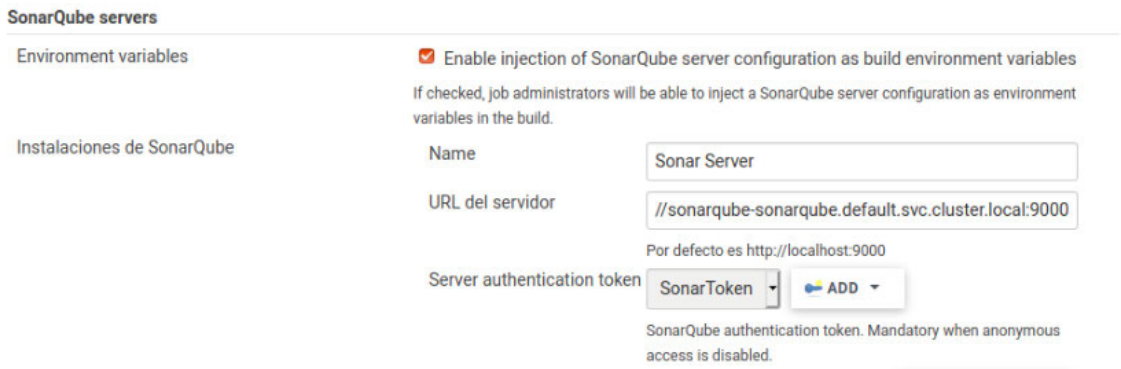


Ilustración 34 - Configuración SonarQube servers

Es necesario incluir en el archivo 'jenkinsfile' una etapa (*stage*) en el *job* 'tfgCI' para el análisis de SonarQube. Se añade una etapa de 'SonarQube Analysis' antes de la que ya existía de 'Compile'. Esta parte del archivo jenkinsfile queda de la siguiente manera:

```
stages{
        stage('SonarQube Analysis'){
                enviroment{
                        scannerHome = tool 'SonarScanner'
                }
                steps{
                        container('maven'){
                                withSonarQubeEnv('Sonar Server'){
                                        sh "${scannerHome}/bin/sonar-scanner"
                                }
                                timeout(time: 10, unit: 'MINUTES'){
                                        waitForQualityGate abortPipeline: true
                                }
                        }
                }
        }
        stage('Compile'){
                steps{
                        container('python'){
                                sh "python prueba.py"
                        }
                }
        }
}
```

Figura 8 - Etapas del pipeline en el jenkinsfile inicial

5.4. Plan de pruebas

Las siguientes pruebas que se describen ayudan a comprobar que se cumplen los requisitos del proceso DevOps y los casos de uso que surgen durante la etapa de análisis y diseño de éste.

Esta es la plantilla que se utiliza para definir cada prueba:

ID de prueba	
Precondición	
Acción	
Postcondición	
Requisito cubierto	
Caso de uso cubierto	

Tabla 28 - Plantilla de especificación de pruebas

Siendo cada uno de los apartados:

- **Id de la prueba**: Identificador inequívoco de la prueba. Formato: PRUEBA-XX, empezando desde la prueba 01.
- **Precondición**: El estado y condiciones que deben darse antes de que se realice la acción.
- **Acción**: Los pasos que se realizan durante la prueba.
- **Postcondición**: Estado en el que se queda el sistema una vez realizada la prueba.
- **Requisito cubierto**: Requisito, o requisitos, a los que se asocia la prueba.
- **Caso de uso cubierto**: Caso de uso que se comprueba con esta prueba.

Las pruebas no sólo sirven para mostrar que el funcionamiento del sistema es el esperado con respecto al proceso DevOps diseñado, sino que puede verse cuál es la manera de utilizar el sistema una vez es configurado.

ID de prueba	*PRUEBA-01 – Iniciar el sistema*
Precondición	El sistema está configurado en el equipo.
Acción	Lanzamos en la consola de comandos el comando de levantamiento del sistema.
Postcondición	El sistema se inicia y es funcional.
Requisito cubierto	RSC-01
Caso de uso cubierto	CU-01, CU-02, CU-03

Tabla 29 - PRUEBA – 01 – Iniciar el sistema

Esta prueba puede comprobarse en la Ilustración 15, insertada anteriormente en este documento.

ID de prueba	*PRUEBA-02 – Automatización de análisis*
Precondición	El sistema está configurado en el equipo y levantado.
Acción	Se hace push de código nuevo sobre el repositorio remoto.
Postcondición	Se sube correctamente el código al repositorio remoto. Se lanza la automatización de Jenkins Se pueden ver los resultados del análisis en SonarQube Se ven propuestas de soluciones a los problemas encontrados
Requisito cubierto	RDB-01 | RDIC-01 | RDFC-01 | RDFC-02 | RDFC-03 | RDFC-04 | RDFC-05
Caso de uso cubierto	CU-01 | CU-02 | CU-03

Tabla 30 - PRUEBA – 02 – Automatización de análisis

En las siguientes capturas de pantalla se refleja el procedimiento que se sigue para realizarla, y el resultado satisfactorio de ésta.

Ilustración 31 - Pruebas - Push a repositorio remoto

Se realiza un commit con el mensaje "prueba fallo master", sobre la rama master. Se realiza un push y para ello Gitea solicita las credenciales del usuario de Gitea.

Ilustración 35 - Pruebas - Automatización en Jenkins

Se comprueba que se lanza el job sobre master. A continuación, consultar las fases del pipeline:

Ilustración 36 - Pruebas - Fases del pipeline de la rama master

Al ver la salida por consola de esta construcción, se observa lo siguiente:

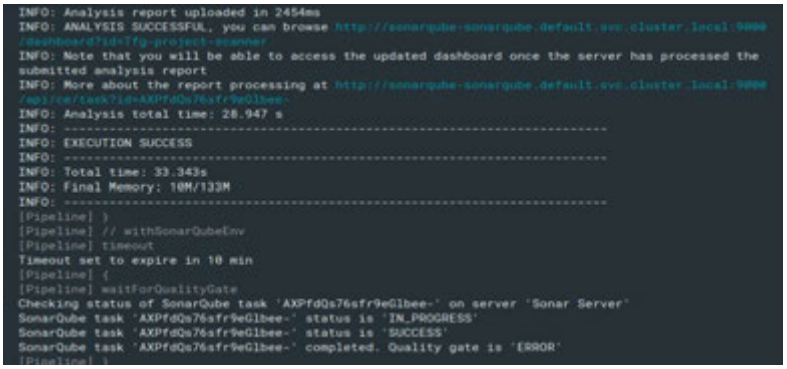


Ilustración 37 - Pruebas - Análisis correcto y no pasa los umbrales de calidad

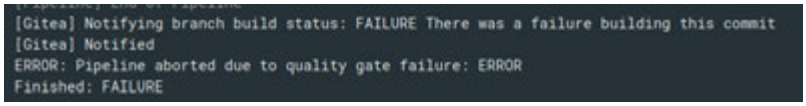


Ilustración 38 - Pruebas - Se aborta el job debido a no pasar los niveles de calidad

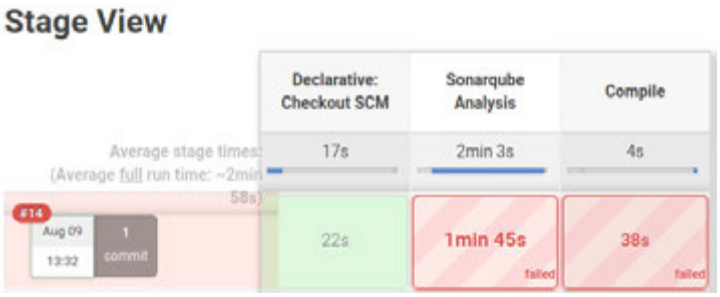


Ilustración 39 - Pruebas - Pipeline fallido condicionado

Se ve que la ejecución es correcta, pero que el pipeline falla debido a que no se pasan los niveles de calidad configurados.

Es posible consultar en SonarQube cuáles son los fallos que abortan la compilación por bajar la calidad del código. A su vez, la plataforma sugiere soluciones que cumplen con las reglas de calidad para solucionar los fallos de la última modificación sobre el código.

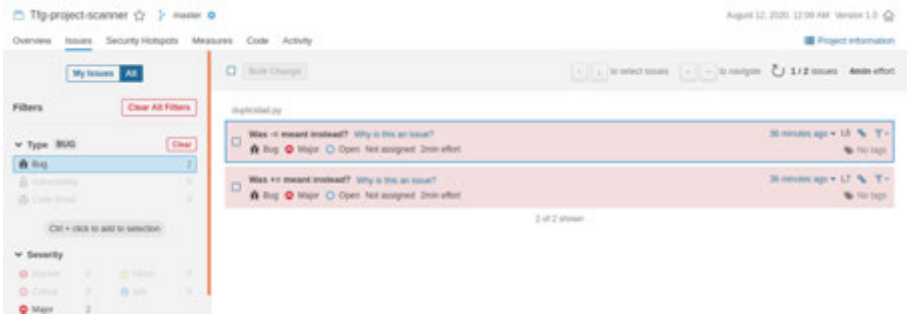


Ilustración 40 - Pruebas - Informe de fallos en Sonar

ID de prueba	*PRUEBA-03 – Compilación condicionada*
Precondición	El sistema está configurado en el equipo y levantado.
Acción	Se modifican los umbrales de calidad Se añade un perfil nuevo de quality gates en SonarQube Se lanza desde jenkins de nuevo el pipeline de la rama master
Postcondición	Se lanza el pipeline con éxito Se visualizan los resultados en SonarQube
Requisito cubierto	RDIC-03 | RDFC-01 | RDFC-02 | RDFC-04
Caso de uso cubierto	CU-2 | CU-3

Tabla 31 - PRUEBA – 03 – Compilación condicionada

En las siguientes capturas, se ven los pasos seguidos para la realización de la prueba, y el resultado satisfactorio de ésta.

- Crear una nueva Quality Gate con los parámetros que se crean convenientes. En este caso, crear uno muy permisivo para poder mostrar en la prueba la flexibilidad de la herramienta.
- Posteriormente, en la configuración del proyecto de SonarQube se le asocia este nuevo perfil de calidad que se ha creado para que comience a aplicarse el nuevo, y no uno por defecto.
- Lanzar la ejecución de Jenkins, y ver que se realiza correctamente, y que, a pesar de los bugs encontrados en el análisis, se realiza la compilación y el resultado es de éxito.

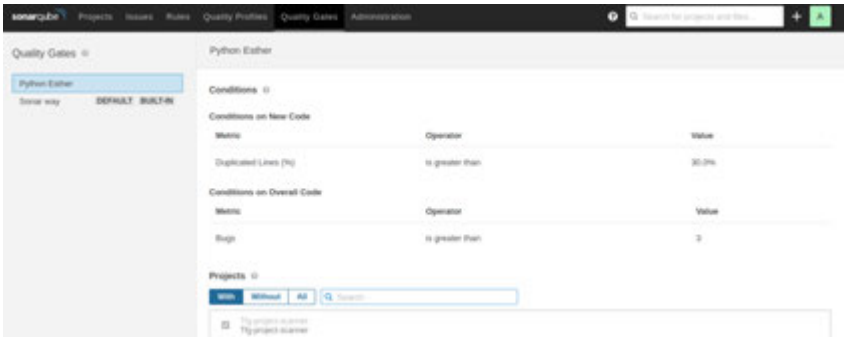


Ilustración 41 – Pruebas - Nuevo quality gate SonarQube

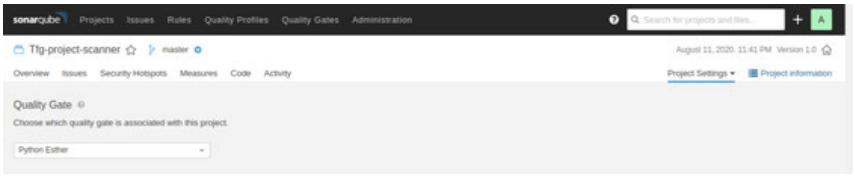


Ilustración 42 - Pruebas – Configurar nuevo quality gate en el proyecto

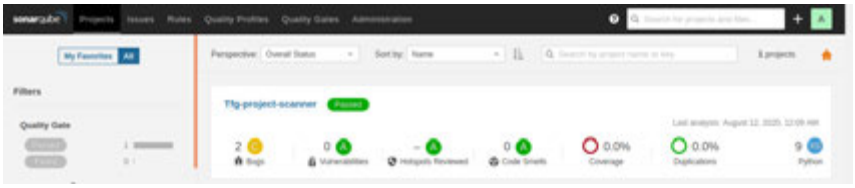


Ilustración 43 - Pruebas - Mismos bugs, pero con análisis favorable

Ilustración 44 - Pruebas – Distinto resultado del mismo código

Es interesante fijarse en que, al relajar los umbrales de calidad y aplicar menos reglas en el análisis, el tiempo de análisis en el pipeline de jenkins se reduce de 1 minuto 45 segundas, a 49 segundos., es decir a un 46% del tiempo que tarda en la anterior ejecución.

ID de prueba	*PRUEBA-04 – Automatización multi rama*
Precondición	El sistema está configurado en el equipo y levantado.
Acción	Cambiar de rama Hacer push del código sobre la rama develop
Postcondición	Se lanza el pipeline con éxito No hay análisis del código sobre la rama develop
Requisito cubierto	RDB-01 | RDB-02 | RDIC-02
Caso de uso cubierto	CU-01

Tabla 32 - PRUEBA-04 – Automatización multi rama

Para mostrar la diferencia de automatizaciones en función de en qué rama del repositorio remoto se realiza la acción, en el jenkinsfile de la rama Develop sólo crear una traza que pinte el mensaje "Hello Develop", y no lanza análisis de código.

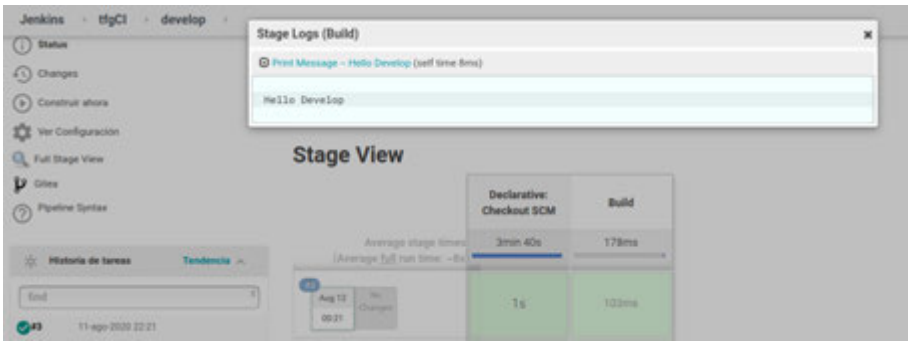


Ilustración 45 - Pruebas - Pipeline de Develop simple

Una métrica de calidad del proceso destacable, no de calidad del código, que se puede estudiar con la información que proporciona Jenkins por defecto es la "salud" de las ejecuciones de las interacciones sobre las distintas ramas. En este caso, por ejemplo, indica lo siguiente:

Ilustración 46 - Pruebas – Métrica de calidad del proceso BSBF

Sólo el 20% de las últimas ejecuciones sobre la rama master son exitosas. Es una salud muy pobre, pero comprensible dado que se realizan muchas pruebas durante la configuración del proyecto, y muchas no son correctas. Pero para un desarrollador que use el sistema, es un dato muy a tener en cuenta cuando está trabajando. A esta métrica, que sería una ratio construcción con éxito (Build Success) frente a construcción fallida (Build Failure), podríamos nombrarla justo con esas siglas del inglés: **BSBF.** Esta métrica indicaría si se está aplicando correctamente el proceso DevOps o si, por el contrario, no se está entendiendo el funcionamiento y el propósito de aplicarlo.

6. PLANIFICACIÓN Y PRESUPUESTO

En este capítulo se procede a detallar la planificación que sigue para ejecutar el proyecto de fin de grado, plasmado en un Diagrama de Gantt. Una vez indicada la planificación, se pasa a desglosar el presupuesto de los recursos invertidos en el proyecto y cuál es el coste del desarrollo de este proyecto.

6.1. Planificación

Dado que se trata de un año atípico, la planificación se ha modificado a lo largo del curso.

Debido a que se compagina la realización del proyecto con un puesto de trabajo como consultora, se opta por la tercera convocatoria de presentación del trabajo de fin de grado. Se selecciona el tema sobre el que trata este proyecto en octubre de 2019, y el periodo de entrega finaliza. para la tercera convocatoria, el día 7 de septiembre de 2020. Se cuenta con 10 meses para la realización del proyecto, pero de trabajo efectivo se puede reducir a una planificación de 4 meses. La planificación termina antes de la entrega de la documentación, por si se produce algún cambio significativo, no tener problemas de tiempo para realizar las modificaciones.

A continuación, en la siguiente página, el diagrama de Gantt que muestra la planificación que se sigue durante el proyecto:

Ilustración 47 - Diagrama de Gantt del proyecto

Se puede observar que la tarea de documentación es la tarea más larga. En esencia, acompaña durante toda la realización del proyecto. Desde la creación de la estructura, hasta las modificaciones pertinentes. Acompaña desde principio, a fin.

Siguiendo con la introducción de la documentación, resulta sencillo identificar las motivaciones dado que el tema del proyecto surge a partir de un interés personal, y no lleva apenas tiempo.

Con respecto al Estado del arte, es destacable que lleva bastante más tiempo del que en un principio se estima. Esto se debe a que, al ser un tema popular, sintetizar toda la información encontrada sobre el tema y que el resultado fuese completo requiere mayor dedicación de lo esperado.

Para poder realizar la implementación, es necesario invertir suficiente tiempo en las tareas de análisis y diseño. En algunos momentos, una vez se comienza la implementación del sistema, se vuelve a retomar la tarea de diseño dado que, en función de cómo avanza la implementación, se encuentran mejoras que se pueden aplicar y que animan a modificar el diseño para reflejarlas.

Las pruebas, por otra parte, se plantean durante el análisis del sistema que se implementa en este trabajo, pero se realizan una vez está terminado el sistema, aparte de las realizadas cuando se están

configurando todos los componentes. Se quiere dedicar el tiempo suficiente para comprobar que el funcionamiento del sistema es el deseado.

6.2. Presupuesto

En este apartado se desglosan los costes que se derivan de la realización de este proyecto. Como costes directos se plantean los costes de recursos humanos y los costes físicos, licencias y dispositivos, y como costes indirectos, los derivados de la luz, agua e internet durante la realización del proyecto.

6.2.1. Recursos humanos

Para la estimación del salario, se escoge como referencia el rango de salarios que hay actualmente en la empresa en la que se desarrolla el puesto de consultora, y se hace una media con los sueldos que indica una calculadora de sueldos online (randstad, 2020), para el puesto de desarrollador y analista.

El coste del cálculo por hora se hace dividiendo el salario bruto, por el número de horas laborales de un mes, cuyas semanas cuentan con 40 horas laborales. Es decir, una media de 160 horas al mes.

Perfil	Salario bruto	Salario / hora	Horas	Total
Analista de sistemas	2387€	14,92€	216	3222,72€
Desarrollador tester DevOps	2090€	13,06€	296	3865,76€
				7088,48€

Tabla 33 - Tabla de costes de recursos humanos

6.2.2. Costes físicos

En este caso, se referencian los costes amortizables. Los equipos utilizados, y las licencias de software de los programas que se utilizan durante el desarrollo del proyecto.

Para calcular los costes de amortización de cada elemento, se toma como referencia la tabla actual que proporciona la agencia tributaria (Agencia Tributaria, 2020). Una vez calculado el coste de amortización por mes dedicado al proyecto, se refleja en el total.

Producto	Precio	Tiempo máximo de amortización (años)	Cantidad amortización al mes	Amortización para el proy.
Portátil HP Envy 14 14-1195ea	1599€	10	13,32€	53,3€
Portátil Macbook Air	1.061,17€	10	8,84€	35,37€
Microsoft Office 365 Personal	69€/año	1	5,75€	23€
Minikube	0€	-	-	-
Virtual Box	0€	-	-	-
				111,67€

Tabla 34 - Tabla de costes físicos

6.2.3. Costes indirectos

Por último, deducir los gastos indirectos que se producen mientras se realiza el proyecto, tal como luz, internet, y agua que se han gastado. Como el tiempo efectivo dedicado al trabajo son aproximadamente cuatro meses, se multiplica la mensualidad de dichos gastos por los cuatro meses.

Gasto	Cuota mensual	Total en el proyecto
Luz	36€	144€
Internet	50€	200€
Agua	17€	68€
		412€

Tabla 35 - Tabla de costes indirectos

6.2.4. Resumen del presupuesto

En la tabla a continuación, mostramos un resumen de todos los costes derivados del trabajo realizado:

Coste	Total invertido
Recursos humanos	7088,48€
Costes físicos	111,67€
Costes indirectos	412€
TOTAL DEL PROYECTO	**7612,15€**

Tabla 36 - Tabla resumen de costes

No se tiene en cuenta si puede reportar beneficios, dado que es un proyecto realizado con el fin de entregar un trabajo de calidad para finalizar el grado, y no se plantea explotarlo desde un punto de vista comercial. Las cantidades indicadas anteriormente incluyen el IVA.

7. MARCO LEGAL Y ENTORNO SOCIO-ECONÓMICO

En este apartado se explican cuales son las leyes o reglamentos aplicables a este trabajo, así como las licencias necesarias para las herramientas que componen el sistema. Una vez analizado esto, son explicados los beneficios que se observan derivados de la aplicación de la filosofía DevOps en entornos de trabajo reales.

7.1. Marco legal

En cuanto aplicaciones utilizadas, son programas de código abierto (open source). Esto significa que permiten acceso a su código fuente, que facilita su uso y modificación para el resto de desarrolladores, no solo los creadores de dichas aplicaciones. Están publicadas de forma gratuita y, por lo tanto, no requieren adquirir una licencia para poder utilizarlos o modificarlos.

Algunas de las aplicaciones que conforman el sistema, requieren autenticarse o introducir datos para crear un usuario. Estas aplicaciones son susceptibles a utilizar datos de carácter personal o sensibles, por lo que hay que tener en cuenta tanto la Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales (LOPD), como el Reglamento General de Protección de Datos (Reglamento (UE) 2016/679 Del Parlamento Europeo Y Del Consejo).

Ley Orgánica de Protección de Datos Personales y garantía de los derechos Digitales:

> *"Artículo 2. Ámbito de aplicación de los Títulos I a IX y de los artículos 89 a 94.*
>
> *1. Lo dispuesto en los Títulos I a IX y en los artículos 89 a 94 de la presente ley orgánica se aplica a cualquier tratamiento total o parcialmente automatizado de datos personales, así como al tratamiento no automatizado de datos personales contenidos o destinados a ser incluidos en un fichero."* (Jefatura del Estado, 2018)

Aunque se introduzcan datos, es un sistema local, por lo que los datos personales que se puedan llegar a introducir no se incluyen en ningún fichero, ni circulan en ninguna plataforma de internet. Esos datos son almacenados en el equipo en el que está instalado el sistema únicamente, y sólo el usuario del sistema tiene acceso a ellos, ni son tratados por terceros.

Reglamento General de Protección de Datos (UE)

"Artículo 1. Objeto

1. El presente Reglamento establece las normas relativas a la protección de las personas físicas en lo que respecta al tratamiento de los datos personales y las normas relativas a la libre circulación de tales datos."

Como se indica con la Ley Orgánica de Protección de Datos (Parlamento Europeo, Consejo de la Unión Europea, 2016), los datos introducidos en las aplicaciones del sistema no son tratados de ninguna manera, y es un sistema local que no está conectado a la red.

7.2. Entorno socio-económico

La aplicación de la filosofía DevOps o la utilización de un sistema de integración continua en equipos de desarrollo trae consigo numerosos beneficios, como se puede consultar en múltiples fuentes (Davis, 2018) (New Relic, s.f.).

En concreto, este proyecto, ayuda al desarrollador a tener una mejor visión de la calidad del código que él mismo está desarrollando, y utilizar esa mejora en la calidad de lo que desarrollar ayuda a sentirte motivado. Además de tener esa visión, ayuda con la familiarización a las herramientas que suelen componer los entornos de integración o despliegue continua. Esto contribuye a una mejor adaptación en los distintos proyectos de desarrollo en los que en un futuro participe el desarrollador, dado que es cada vez más común que en el mundo laboral te encuentres utilizando esta filosofía aplicada a los proyectos de desarrollo.

Del informe de State of Devops, de 2019 (Dr. Nicole Forsgren, Dr Dustin Smith, 2019), es posible resumir los beneficios de la aplicación de esta filosofía en los siguientes puntos:

- La entrega continua, rápida, fiable y segura de código es la clave para la transformación digital y el desempeño exitoso de una organización, o compañía. Este buen desempeño incluye a nivel de beneficios, productividad y satisfacción por parte del cliente.
- La mayor productividad derivada de esta filosofía puede impulsar mejoras para la conciliación vida-trabajo, y reducir el agotamiento de los empleados, y permite que las compañías puedan invertir en respaldar ese equilibrio.
- Comparando equipos que están en la élite del rendimiento, con los equipos de menor rendimiento, se obtienen datos muy ilustrativos sobre los beneficios que trae la aplicación de la integración continua, como se puede ver en esta imagen:

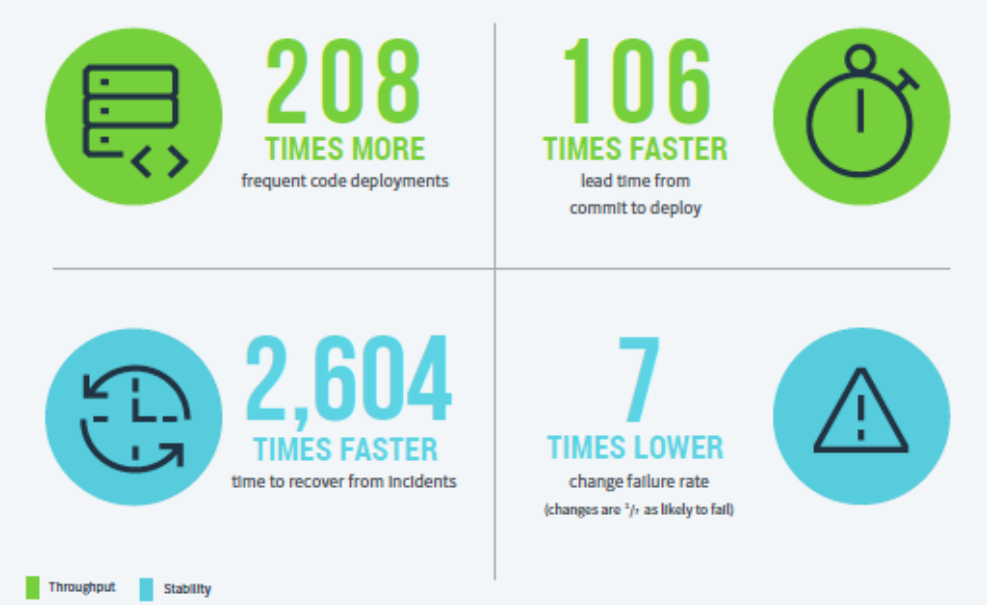


Ilustración 48 – Comparativa rendimiento de equipos DevOps – no DevOps (Dr. Nicole Forsgren, Dr Dustin Smith, 2019)

Los equipos de alto rendimiento despliegan 208 veces más, 106 veces menor el tiempo entre commit y despliegue, recuperación tras una incidencia 2604 veces más rápidamente, y con una ratio 7 veces menor de modificación/error.

- DevOps trae consigo una cultura psicológica enfocada a la seguridad. Se demuestra que los equipos de alto rendimiento adoptan una cultura de confianza y seguridad, que les permite tomar riesgos menores y más calculados, comunicarse sin miedo, y centrarse en la creatividad. Los miembros del equipo se implican más con el trabajo que hacen, y los tiempos de entrega se cumplen en la mayoría de los casos.
- Contar continuamente con el *feedback* (realimentación) de los clientes o usuarios, ayuda a los equipos a reaccionar más rápidamente para ofrecerles soluciones de mayor calidad y que proporcionen más beneficios a los usuarios.

8. CONCLUSIONES Y TRABAJO FUTURO

En este apartado, se explican las conclusiones a las que se llegada basadas en los objetivos marcados al comienzo de este proyecto. Y a continuación de las conclusiones, los pasos que pueden darse en el futuro para mejorar el sistema creado.

8.1. Conclusiones

El primer objetivo principal que se marca es diseñar e implementar un sistema DevOps. No siendo un sistema de entrega continua, sí puede considerarse un sistema de testing continuo e integración continua. Se considera que se ha conseguido este objetivo principal, y que se ha desarrollado un producto que puede ser utilizado para mejorar la calidad del código que se desarrolla o entrega a diario. Además, si alguien llega a configurárselo en su equipo, entenderá con mayor facilidad cuales son las operaciones que corren por detrás en los sistemas de integración continua que use en sus futuros proyectos.

El segundo objetivo es investigar sobre la filosofía DevOps, y que resulte más fácil adoptarla porque cada día es más popular en la entrega de software. Se puede afirmar que el conocimiento adquirido sobre este campo ha sido mayúsculo, y, de hecho, esto ha provocado que en el entorno laboral se me haya convertido en la persona de referencia del equipo cuando algún otro miembro tiene alguna duda o problema con los procesos de integración continua del proyecto. Es decir, que no sólo se ha podido comprobar un mayor conocimiento de forma personal sobre el área, sino que el entorno también lo ha percibido.

Los objetivos derivados llevaron a utilizar las herramientas que finalmente se escogieron, para que el sistema fuera multiplataforma y pudiera instalarse en cualquier equipo. Se piensa que el sistema que ha sido diseñado puede adaptarse fácilmente a cualquier proyecto que se realice, incluidas las prácticas de las distintas asignaturas del grado que termino con este trabajo, y que su uso ayudará a adoptar esta metodología que tantos beneficios trae. Y, por último, como se apuntaba en las pruebas, con el uso de este sistema pueden estudiarse no sólo la calidad y la seguridad, sino los tiempos de los distintos procesos (análisis, compilaciones, …).

Finalmente, a titulo personal, añadir que me ha encantado poder elegir este tema para terminar la carrera y aprender tanto sobre este tema, y haber desarrollado un sistema tan práctico y al que puede dársele un uso real y que pueda ayudar a más gente a crecer como desarrolladores.

8.2. Trabajos futuros

Por último, indicar algunos pasos que, por falta de tiempo o limitaciones del hardware disponible, no se han podido implementar pero que pueden añadirle mucho valor al sistema y así mejorarlo.

Añadir un repositorio de artefactos

Es común que, cuando se realiza un cambio sobre el código de determinadas ramas en el repositorio remoto, se almacenen las distintas versiones de los componentes compilados para, posteriormente, poder desplegar en el servidor de aplicaciones una versión en concreto. Se puede así desplegar versiones antiguas de los componentes sin necesitar construir el código de nuevo.

Añadir descargas de dependencias del código

Estas dependencias suelen ser necesarias para compilar los artefactos, cuando son más complejos que los que se han utilizado para el desarrollo del sistema. Al descargar estas dependencias, podrán ser analizadas por la herramienta de análisis de código. Es necesario porque estas dependencias externas pueden estar creando vulnerabilidades en las aplicaciones desarrolladas sin saberlo.

Crear plantillas para los distintos ciclos de vida

No son las mismas etapas, ni comandos, que se realizan en un pipeline para todos los tipos de aplicaciones. Es decir, la construcción y análisis de una aplicación Maven (es decir, Java), no tiene las mismas etapas que el de una aplicación Node (JavaScript), o Python. Se pueden crear plantillas, que almacenaríamos en un repositorio aparte, donde estarían los distintos Jenkinsfile genéricos para todos estos tipos de aplicaciones.

Añadir un servidor para desplegar las aplicaciones

Si se quisiera poder desplegar de forma automática los artefactos que compilamos haciendo uso de Jenkins, podríamos instalar un servidor local, por ejemplo, Tomcat o Glassfish. Ahorrando así el tiempo de despliegue al no tener que hacerse de forma manual.

Estudio de profundidad de las métricas que se reflejan tanto en Jenkins como en SonarQube

Posiblemente, lo que mayor valor puede aportar, es hacer uso de las métricas (tiempos, errores por commit, calidad, seguridad) que ofrecen tanto Jenkins como SonarQube. Atender a esta información y aprender de ella es lo que más puede ayudar a crecer como desarrolladores.

9. BIBLIOGRAFÍA

Agencia Tributaria., 2020. Tabla De Amortización Simplificada. April 6, 2020, Available from: https://www.agenciatributaria.es/AEAT.internet/Inicio/Ayuda/Manuales__Folletos_y_Videos/Manuales_practicos/_Ayuda_Folleto_Actividades_economicas/3__Impuesto_sobre_la_Renta_de_las_Personas_Fisicas/3_5_Estimacion_directa_simplificada/3_5_4__Tabla_de_amortizacion_simplificada/3_5_4__Tabla_de_amortizacion_simplificada.html.

AUTHORS, H., s.f. Helm Repo List. Sin fecha, Available from: https://helm.sh/docs/helm/helm_repo_list/.

AWS, A., s.f. ¿Qué Es DevOps? Sin fecha. Available from: https://aws.amazon.com/es/devops/what-is-devops/.

AWS, A., s.f. ¿Qué Es La Integración Continua? Sin fecha. Available from: https://aws.amazon.com/es/devops/continuous-integration/.

BENITEZ, A., 2018. Introducción a Kubernetes y Minikube. Marzo, 2018, Available from: https://blog-es.mimacom.com/introduccion-kubernetes-minikube/.

BUCHANAN, I., 2019. Las Metodologías Ágiles y DevOps: ¿amigos o Enemigos? 2019, Available from: https://www.atlassian.com/es/agile/devops.

COSIO, L., 2019. ¿Cómo Funcionan Los Contenedores De Software? Febrero 15, 2019, Available from: https://www.inbest.cloud/comunidad/como-funcionan-los-contenedores-de-software.

DAVIS, B., 2018. DevOps Dollars: Why there's Big Money in Fast Software Development. May 17, 2018, Available from: https://www.forbes.com/sites/forbestechcouncil/2018/05/17/devops-dollars-why-theres-big-money-in-fast-software-development/#20db183061e6.

DevOps Practitioners., s.f.. Periodic Table of DevOps Tools. Sin fecha, Available from: https://xebialabs.com/periodic-table-of-devops-tools/.

Dr. Nicole Forsgren, Dr. Dustin Smith, Jez Humble, Jessie Frazelle., 2019. Accelerate - State of DevOps 2019.

Google Cloud., 2020. Tecnología De DevOps: Arquitectura. [Online].2020, may 29, Available from: https://cloud.google.com/solutions/devops/devops-tech-architecture?hl=es-419.

GRAZAS, J., 2014. Aprende a Implantar Integración Continua Desde Cero, ¿Por Qué Integración Continua? Agosto 8, 2014, Available from: https://www.javiergarzas.com/2014/08/implantar-integracion-continua.html.

Jefatura del Estado., 2018. Ley Orgánica 3/2018, De 5 De Diciembre, De Protección De Datos Personales y Garantía De Los Derechos Digitales. 6 de Diciembre de 2018, Available from: https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf.

Juan Quijano., 2018. El Ciclo De DevOps, Una Guía Para Iniciarse En Las Fases Que Lo Componen. 21 Junio 2018, Available from: https://www.genbeta.com/desarrollo/el-ciclo-de-devops-una-guia-para-iniciarse-en-las-fases-que-lo-componen.

KABIR, A., 2017. Continuous Integration: A "Typical" Process. September 6, 2017, Available from: https://developers.redhat.com/blog/2017/09/06/continuous-integration-a-typical-process/?extIdCarryOver=true&sc_cid=701f2000001OH7EAAW.

KAPADIA, M., 2016. Designing a DevOps Strategy: A Balanced Scorecard Framefork. February 18, 2016, Available from: https://devops.com/designing-devops-strategy-balanced-scorecard-framework/.

kubernetes.io., s.f.. Instalación y Configuración De Kubectl. Sin fecha, Available from: https://kubernetes.io/es/docs/tasks/tools/install-kubectl/.

learnitguide.net., 2018. How to Configure Jenkins Mail Notification on Build Failures. August 11, 2018, Available from: https://www.learnitguide.net/2018/08/how-to-configure-jenkins-mail.html.

Microsoft Azure., s.f.. ¿Qué Es DevOps? Sin fecha, Available from: https://azure.microsoft.com/es-es/overview/what-is-devops/.

Mike's Software Blog., 2019. How to Integrate Gitea and Jenkins. May 30, 2019, Available from: https://mike42.me/blog/2019-05-how-to-integrate-gitea-and-jenkins.

MOY, E., 2019. DevOps Tools of the Trade. September 28, 2019, Available from: https://www.osolabs.com/blog/devops-tools-of-the-trade/.

New Relic., s.f.. The Benefits of DevOps. Sin fecha, Available from: https://newrelic.com/devops/benefits-of-devops.

OCHOA, M., 2018. DevOps Tools by Excentia, Sept. 14, 2018.

Parlamento Europeo, Consejo de la Unión Europea., 2016. Reglamento General De Protección De Datos. 27 de abril de 2016, Available from: https://www.boe.es/doue/2016/119/L00001-00088.pdf.

Philippe Kruchten, 1995. Planos Arquitectónicos: El Modelo De “4+1” Vistas De La Arquitectura Del Software, Noviembre 1995.

randstad., 2020. Calculadora Salarial. Available from: https://www.randstad.es/calculadora-salarial/candidatos-resultado.

RedHat., s.f.. ¿Qué Es Un Clúster De Kubernetes? Sin fecha, Available from: https://www.redhat.com/es/topics/containers/what-is-a-kubernetes-cluster.

RedHat., s.f. ¿Qué Son La integración/distribución Continuas (CI/CD)? Sin fecha, Available from: https://www.redhat.com/es/topics/devops/what-is-ci-cd.

SOLUTIONDOT, T.B., 2018. Difference between DevOps and Continuous Integration. May 07,2018, Available from: https://solutiondots.com/blog/technology-blog/difference-devops-continuous-integration/.

TEAM, C., s.f.. DevOps: Arquitectura Monolítica Vs Microservicios. Sin fecha, Available from: https://www.chakray.com/es/devops-arquitectura-monolitica-vs-microservicios/.

TEAM, K., s.f.. Installing Kubernetes with Minikube. Sin fecha, Available from: https://kubernetes.io/docs/setup/learning-environment/minikube/.

Wikipedia.,2020. Log (informática). May 12, 2020, Available from: https://es.wikipedia.org/wiki/Log_(informática)

kubernetes io. s.f.. Pods. Sin fecha, Available from: https://kubernetes.io/es/docs/concepts/workloads/pods/pod/#qué-és-un-pod

Node.js., s.f. Acerca de Node.js, Sin fecha. Available from: https://nodejs.org/es/about/

Codeglia, A., 2018. Metodología lean: ¿qué es y cómo aplicarla en tu negocio? May 5, 2018, Available from: https://blog.hotmart.com/es/metodologia-lean/

Data Management., 2019. Webhooks: Cómo interconectar aplicaciones web February 26, 2019,.https://www.analiticaweb.es/webhooks-interconectar-aplicaciones-web/

Atlassian., 2020. Flujo De Trabajo De Gitflow. January 1, 2020, Available from: https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow.

CAÑAS, S., 2020. 5 Softwares De Control De Versiones. February 11, 2020, Available from: https://www.drauta.com/5-softwares-de-control-de-versiones.

CLOUD, G., s.f. Lineamientos Para Crear Clústeres Escalables. Sin Fecha, Available from: https://cloud.google.com/kubernetes-engine/docs/best-practices/scalability.

FONG, J., 2018. Are Container Replacing Virtual Machines? August 30, 2018, Available from: https://www.docker.com/blog/containers-replacing-virtual-machines/.

Kubernetes.io., s.f.. Objetos De Kubernetes. Sin fecha, Available from: https://kubernetes.io/es/docs/concepts/#objetos-de-kubernetes.

PROYECTOS, W., 2020. Control De Versiones. August 1, 2020, Available from: https://es.wikipedia.org/wiki/Control_de_versiones.

SHIPAHI, T., Alfred, Neokyle and BURKET, K., 2019. Diferencia Entre Los Tipos De Servicio ClustIP, NodePort y LoadBalancer En Kubernetes. January 23, 2019, Available from: https://www.it-swarm.dev/es/kubernetes/cual-es-la-diferencia-entre-los-tipos-de-servicio-clusterip-nodeport-y-loadbalancer-en-kubernetes/828776495/.

TYCZYNSKI, W., 2019. Kubernetes Scalability and Performance SLIs/SLOs. October 17, 2019, Available from: https://github.com/kubernetes/community/blob/master/sig-scalability/slos/slos.md.

SonarSource S.A., 2020. *Quality Profiles*. July 1, 2020, Available from: https://docs.sonarqube.org/latest/instance-administration/quality-profiles/.

Sicilia, Miguel Ángel., 2012. Métricas del Mantenimiento de Software, October 16,2012, Available from:https://cnx.org/exports/a0484c7c-5b86-4ef2-9f7e-fb6bb11dfe59@9.1.pdf/m%C3%A9tricas-del-mantenimiento-de-software-9.1.pdf

Norma ISO25000, 2019. ISO25010 – Fiabilidad, 2019: Available from: https://iso25000.com/index.php/normas-iso-25000/iso-25010/24-fiabilidad

10. DICCIONARIO

Máquina Virtual: En informática, una máquina virtual es un software que simula un sistema de computación y puede ejecutar programas como si fuese una computadora real.

Repositorio de código remoto: Un repositorio de código es un lugar donde el código de una aplicación, de un programa cualquiera está almacenado y desde donde se puede distribuir.

Clúster: En el contexto de este proyecto, se denomina clúster a un grupo de sectores ubicados en espacios contiguos, que se constituyen como la unidad de almacenamiento más pequeña en el disco en cuestión.

Despliegue: son todas las actividades que hacen que un sistema de software esté disponible para su uso en un servidor de aplicaciones.

Testing: Proceso de probar el software y su funcionalidad.

Automatización: En este contexto, es el conjunto de procesos informáticos que operan con mínima o nula intervención del ser humano.

CPU: Es la unidad central de procesamiento dentro de un ordenador, o un elemento programable, que interpreta las instrucciones de las aplicaciones informáticas.

RAM: Es la memoria de trabajo del ordenador, para el sistema operativo y la mayoría de las aplicaciones que corran en el ordenador.

Consola de comandos: Es la interfaz que se utiliza para ejecutar instrucciones del sistema operativo de manera más sencilla.

Logs: "En informática, se usa el término registro, log, para referirse a la grabación secuencial en un archivo o en una base de datos de todos eventos que afectan a un proceso particular (aplicación, actividad de una red informática, etc.). De esta forma constituye una evidencia del comportamiento del sistema." (Wikipedia, 2020)

Compilación: Es el proceso mediante el cual se traduce el código fuente de una aplicación para generar un objeto ejecutable.

Artefacto: En el contexto de este trabajo, un artefacto es cualquier módulo de software generado tras la compilación de un código fuente.

Token: Es el código generado, basado en una cadena de caracteres, que permite la autenticación segura en un sistema fácilmente.

Pod: El objeto más sencillo de Kubernetes. Es un grupo de uno o más contenedores con almacenamiento/red compartidos, y unas especificaciones de cómo ejecutar los contenedores (kubernetes io, 2020)

ITIL: Es un acrónimo de las siglas en inglés de Biblioteca de Infraestructura de Tecnologías de Información. Es una metodología de gestión, que engloba buenas prácticas de organización de recursos gestionados por los departamentos de TI (tecnologías de la información) de las compañías.

Scrum: Es una metodología de trabajo que ayudan a aplicar procesos estandarizados de forma recursiva y colaborativa entre miembros de un equipo, para obtener mejores resultados en los proyectos que se lleven a cabo.

Node.js: Es un entorno de ejecución de JavaScript orientado a eventos asíncronos, está diseñado para crear aplicaciones network escalables. (Node js, 2020)

Maven: Apache Maven es un software de gestión y construcción de proyectos basados en el lenguaje de programación java.

Java: Lenguaje de programación y una plataforma informática.

Python: Lenguaje de programación.

Evolutivo: En este contexto, un evolutivo es un desarrollo de mejora sobre un componente ya existente en una aplicación.

Lean: "Lo principal de la metodología lean es evitar desperdicios, usar al máximo los recursos ya disponibles y mejorar constantemente los procesos para obtener cada vez resultados más eficientes." (Codeglia, 2018)

TFG: Trabajo de fin de Grado.

Webhook: Es un método, más sencillo que hace una API, que permite comunicar en tiempo real, a través de HTTP, dos aplicaciones o servicios web cuando se cumplan ciertos criterios. (Data Management, 2019)

ANEXO I – DEMOSTRACIÓN DE COMPETENCIAS EN INGLÉS

INTRODUCTION: MOTIVATION AND PROJECT GOALS

In this first chapter of the document that constitutes the end of studies project, we briefly put both the project and documentation into context.

In the first place, we will describe the motivation that has led me to want to carry out the project on this subject, and after that, the goals that I tried to achieve in it.

1. Motivation

Currently, and for a little over a year and a half, I am part of a development team of corporate's tools of a very important client, that belonges to the telecommunications context. One of the characteristics of the project and the team is that we are not just developers or analysts, but we also act as the operations team.

In the last quarter of 2019, we started working with the client's DevOps team, so we started using their methodology and tools for version control of our developments, deployments, and development integrations, quality metrics focused on our code.

This situation led me to want to learn more about DevOps, the different tools, and the concepts of continuous integration and continuous delivery:

- First, because I wanted to know the tools that we began to use as part of our day to day. The possibilities that these tools offered us, as well as finding the reasons that led them to convince us that adapting to this methodology was not only beneficial at the business level, but that it would also be better for us.
- In addition to this, I began to consider the benefits that the design and installation of a local replica of this continuous integration system could bring, so that all team members, whether or not responsible for the component deployments, would know how to use it, and on a daily basis adapt to the new methodology with which we began to work.

So, putting together this new situation and my curiosity about the DevOps world, led me to want to make a small local system that can help in the development tasks of the day-to-day components of the team, both of the current members and of the team members. future members.

2. Project goals

Given the motivation that led me to want to carry out this project, it was possible to set goals that would lead me to do a quality work.

The main goals are listed below:

- Design and implement a DevOps system locally. As much as possible, similar to the system we use in the project I am currently working on professionally.
- Research about DevOps, and thus be able to adopt this methodology which use is increasingly widespread in the world of software production.

Other objectives, derived from the two main ones above:

- Create a local system that is universal. In other words, that can be installed on any computer, regardless of its operating system.
- Study and measure the reduction of time from the moment we develop a component until it can be deployed on a server, regardless of whether it is local or remote.
- Generate a work methodology that is as standardized as possible, which can be applied regardless of the project in which you are working on.
- Get to create a tool that can be used by any developer, if possible, at any level, or even contribute to the fact that in the near future UC3M students can set up a DevOps system for the development of their subjects projects.

SYSTEM DESIGN AND DEVELOPEMENT PROPOSAL

In order to develop a functional and quality local devops system, it has been necessary to carry out an analysis of the needs of that system and a complete documentation of the process.

1. Software scope

As it was mentioned before, the purpose of generating this local devops environment is for developers to become familiar with this methodology, in addition to performing a quality and security analysis of the different versions of their code.

The system will allow, through the use of version control software, to generate the branches that are necessary for the user, and to analyze them all separately.

The system will not deploy components on a server, neither local nor remote.

The advantages derived from the development of this project are that developers who have not yet worked on a project in which continuous integration techniques (or devops), are applied, can use it as a form of day-to-day work. This way, when the developer has to enter a project of these characteristics, they will already have experience and be fluent on them. They are increasingly popular methodologies, so it can be a great accelerator of capabilities for users who use it.

2. Product Features

As general capabilities of the system, we highlight the following components:

- A Kubernetes cluster: Kubernetes is an orchestrator of containers on which the rest of the components of the local integration system will run *(RedHat, s.f.)*.

- A version-control software: With it, we can have different branches with different code in each one of them, on which we can make different analyzes on the code versions registered in them.
- An orchestrator, or automation server, that will help us automate some parts of the software development process, such as the launch of the code quality or safety analysis when we modify and upload new code to our code repository.

- A static code analysis platform, where we can evaluate and visualize the impact that our modifications have on the code in terms of quality or security, since it will have a friendly control panel where we can quickly detect that said impact.

Our system, then, will be a composite of these elements that will allow us to analyze the code automatically as we are developing it.

3. Restrictions

The system can be installed regardless of the operating system that the equipment has, since they are components that support both Linux, Windows or MacOS, the most common operating systems currently.

Code analysis will be possible for programming languages that are currently covered by the platform chosen for this purpose. For some languages, coverage is only available in the paid version of the platform.

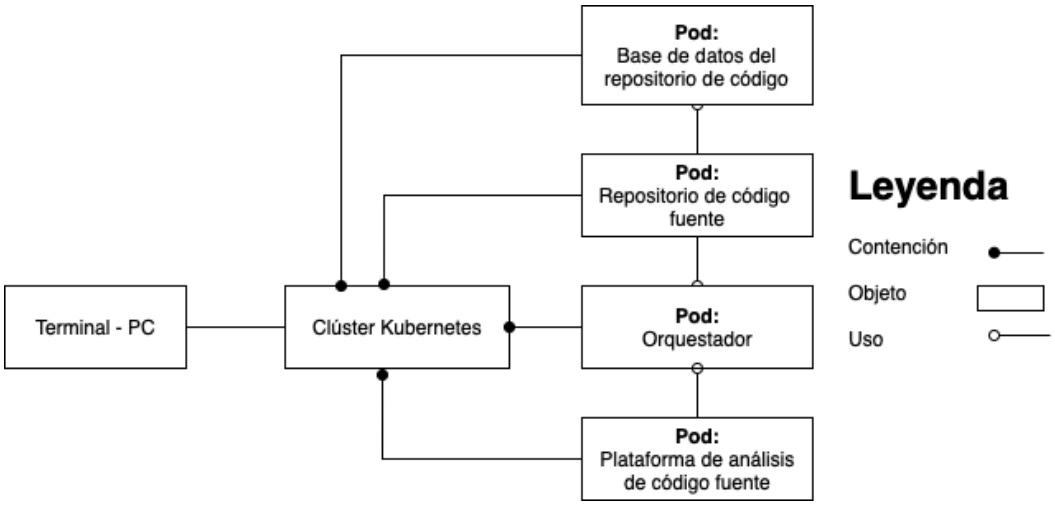
The size and power of the local system will be limited by the processing capacity of the computer where we want to configure it. For the use of SonarQube, the minimum RAM memory that we have needed to achieve a stable system has been 16GB. If we wanted to install it on a computer with less memory, we would have to adapt the features of the system.

This product does not cover the deployment or continuous feedback part of the DevOps software development lifecycle, as we have not configured any servers on which to deploy the builds, but it is possible to do so. With regard to continuous feedback, we do not have any end user or client to suggest us developments on the built product, but with quality and safety analyzes we obtain data to make improvements on the code.

4. Logical view

In this view, an object-oriented design method is used to describe the objects that make up the model. This view primarily asserts the relationship between system components that satisfy the

DevOps process requirements. For graphical representation, only the relevant objects for architecture are taken into account.



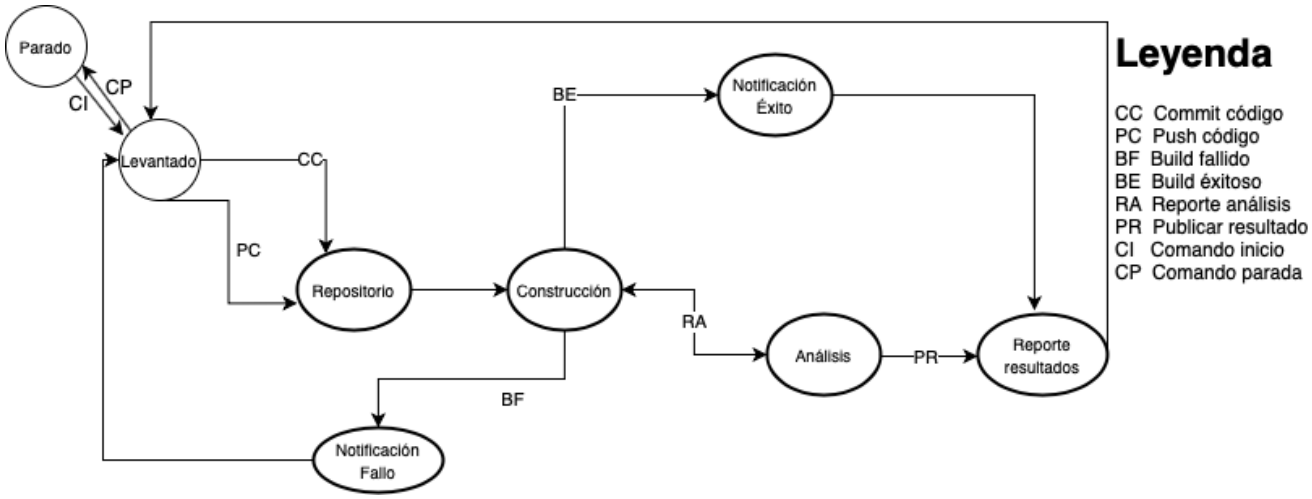
Logical or conceptual view

Explanation of the relationship of components in the diagram:

- Terminal – PC:
  - The equipment on which the system will be installed
  - The container in which the rest of the installed applications will be located is displayed
- Kubernetes cluster:
  - The different applications, pods, that make up this system are installed.
- Database pod:
  - Pod associated with a volume, in which the code repository information is persisted. Therefore, the code repository makes use of it.
- Code repository pod:
  - Deploys the source code repository application.
  - Integrated with the orchestrator application, for it to be able to launch automations.
- Orchestrator pod:
  - Deploys the automation platform application.
- Analysis platform pod:
  - Deploys the static analysis platform application, for code analysis.
  - Integrated with the orchestator application, for it to be able to launch the analyzes automation.

5. Process view

Also known as a run view. In it, we graphically describe the phases that the system can go through, depending on the event that triggers it.



Process, or run. view

Each oval is a phase that can be traversed, as the system has been designed. The three main applications that make up the system can be distinguished, being the source code repository, the construction phase, which is the application that acts as an orchestrator, and analysis and reporting tool, which is the source code analysis platform.

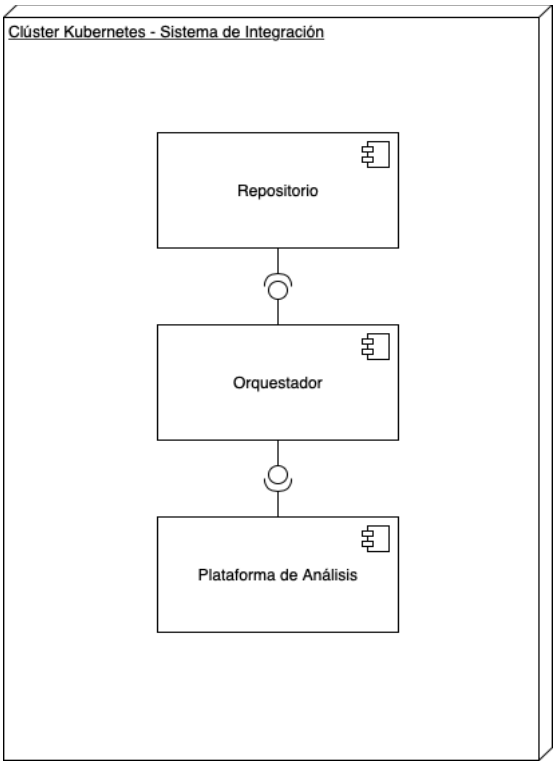
It is possible to observe that two actions can be performed to interact with the code repository, these being a push of the code or a commit of the code.

It is seen that the event that occurs between construction and analysis is bidirectional. This is because from the construction phase the analysis is launched, and the construction can be interrupted depending on the result of the analysis.

Both from the notification of a construction failure (learnitguide net, 2018), and from the publication of the results of the analyzes, the system would go to the initial state, awaiting interaction by the user.

6. Development view

Also known as the implementation view, it is focused on the organization of the modules that make up the system for its development. It can be decomposed into modules or layers, clearly showing the interfaces that each component offers to others for integration.



Development, or implementation. view

In the diagram above, we discussed the system modules that will be installed in the Kubernetes cluster and that will make up our continuous integration system.

The developer user will have a code repository with version control software, an orchestrator software that will carry out the automation actions, and an analysis platform that will carry out the quality and safety analyzes, and which they will be able to access to view the results of such analyzes.

With these three applications, we identify the three components within the system. A component is a modular element that can be substituted in its own context. Its internal elements are not shown, but it has one or more defined interfaces of its own, through which its capabilities can be used.

- The Repository component is in charge of storing the modifications that are made to the source code that the developer needs. It integrates with the Orchestrator component through an interface that it offers, allowing it to collect information about the events carried out in the repository, in order to launch jobs.

- The Orchestrator component is in charge of performing actions automatically based on the event carried out on the code repository, through the so-called Pipelines. It also communicates with the Analysis Platform component, through an interface through which it sends and receives information.

- The Analysis Platform component, in charge of scanning the current code of the code repository and showing the results in its control panel. The analyzes are launched by actions of the Orchestrator component, and the Platform also sends information to the Orchestrator to launch other actions based on the results of the analyzes.

7. Technological infrastructure: Tools used

For the composition of the system, I have chosen the following technologies:

- **Minikube**: It is a reduced and local version of Kubernetes, which using a hypervisor raises a local cluster of a single node, making it act as master and slaves at the same time. It requires fewer Kubernetes resources, and for the system that was built in this project it is more than enough. Its installation is simple, and there is a lot of documentation so its configuration is simple and fast. It can be handled by commands, but a control panel can also be used, which is installed by default, if necessary.

- **Gitea**: It is an open source tool that allows you to host your projects using a development version control system using Git. Gitea was chosen, over other better known ones like GitLab or GitHub, due to prior knowledge and because its known that it has a very wide community of users, so there is a lot of documentation available to make it easier to install it on a local system and integrate it with other applications.

- **Jenkins**: It is a free, open source, automation or continuous integration server. Probably the most used automation server today, so there is a lot of documentation to rely on when you

are going to install it. In addition to the amount of documentation, Jenkins is the orchestrator that we use at work, in the project I am currently working on, as an orchestrator in our continuous integration process, so I was interested in knowing how it works. The diversity of extensions that can be installed to be able to integrate with the rest of the tools that make up the system developed in this degree's final project has also been decisive in choosing Jenkins as orchestrator.

- **SonarQube**: It is a static code analysis platform. Analyze both the quality and the possible vulnerabilities that may exist in our developments. It tells us duplicities, bugs, dead code, what percentage of our code is covered by unit tests. In short, it is a very useful tool for constant improvement of our code. As with Jenkins, Sonar is the tool that I check in at on a daily basis, so I am interested in learning more about it. The other tool that I would have liked to integrate into the system was ELK, but due to the RAM capacity of the computer where I started to build it, it was not possible for me to use it, and finally I chose Sonar.

- **VirtualBox**: It is a virtualizer developed by Oracle Corporation. It is the hypervisor that we will use to build Minikube on it.

CONCLUSIONS AND FUTURE WORK

In this section, I will explain the conclusions that were reached based on the goals that were set at the beginning of this project. And here are the steps that could be taken in the future to improve the system created.

1. Conclusions

The first main goal set was to design and implement a DevOps system. Not being a continuous delivery system, it can be considered a continuous testing system. I believe that I have achieved this main goal, and that I have developed a product that can be used to improve the quality of the code that we develop or deliver on a daily basis. In addition, if someone manages to configure it on their computer, they will understand more easily what are the operations that run behind in the continuous integration systems that they will use in their future projects.

The second main goal was to look into the DevOps philosophy, and to make it easier to adopt it because it is becoming more and more popular in software delivery. I can affirm that my knowledge about this field has grown enormously, and, in fact, this has caused that in my work environment I have become the reference person of the team when another team member has any doubt or problem with continuous integration processes in the project. In other words, not only have I noticed my greater knowledge of the area, but my surroundings have also perceived it.

The derived objectives led me to use the tools that I finally chose, so that the system was universal and could be installed on any computer. I believe that the system that I have designed can be easily adapted to any project that is carried out, including the practices of the different subjects of the degree that I finish with this work, and that its use will help to adopt this methodology that brings so many benefits. And finally, with the use of this system, not only the quality and safety can be studied, but also the times of the different processes (analysis, compilations, ...).

Finally, I would like to add that I was delighted to be able to choose this topic to finish my degree and learn so much about it, and to have developed such a practical system that can be put to real use and that can help more people to grow as developers.

2. Future Work

At last, indicate some steps that I have not been able to implement, due to lack of time, but that I believe can add a lot of value to the system and thus improve it.

Add an artifact repository

It is common that, when a change is made to the code of certain branches in the remote repository, the different versions of the compiled components are stored in order to later be able to deploy a specific version on the application server. You can thus deploy old versions of the components without needing to build the code again.

Add code dependency downloads

These dependencies are usually necessary to compile the artifacts, when they are more complex than the ones we have used for the development of the system. By downloading these dependencies, they can be analyzed by the code analysis tool. It is necessary because these external dependencies may be creating vulnerabilities in our applications without us knowing.

Create templates for different lifecycles

They are not the same steps, or commands, that are performed in a pipeline for all types of applications. That is, the construction and analysis of a Maven application (that is, Java), does not have the same stages as that of a Node application (JavaScript), or Python. Templates can be created, which we would store in a separate repository, where the different generic Jenkinsfiles for all these types of constructions would be.

Add a server to deploy the applications

If we wanted to be able to automatically deploy the artifacts that we compile using Jenkins, we could install a local server, such as Tomcat or Glassfish, and we would save the time of having to do manual deployments.

In-depth study of the metrics that are reflected in both Jenkins and SonarQube

I think that what can bring the greatest value is to make use of the metrics (times, errors per commit, quality, security) that both Jenkins and SonarQube offer us. Attending to this information and learning from it is what will most help us grow as developers.